Jean Engels

HTML 5 et CSS 3

Cours et exercices corrigés

 ISBN : 978-2-212-13400-1

EYROLLES

Table des matières

CHAPITRE 6

CHAPITRE 7

Avant-propos

On ne peut pas encore vraiment qualifier HTML 5 de révolution ; cependant, il marque des avancées dans le domaine de la création de pages web. Avancées en termes de multimédia avec des éléments audio et vidéo natifs affranchissant l'utilisateur des plug-ins habituels. Avancées également, mais peut être moins perceptibles pour le grand public, en termes de sémantique avec l'introduction de nombreux éléments aux rôles prédéterminés qui permettent de structurer précisément les contenus des sites Internet. Son association indispensable avec les feuilles de styles CSS 3 *(Cascading Style Sheets)* rend possible la consultation des pages sur les terminaux les plus divers, allant de l'ordinateur classique à la tablette et au téléphone portable, grâce aux multiples possibilités d'adaptation offertes par CSS.

La conception des sites devant s'effectuer idéalement en deux phases (contenu et présentation), cet ouvrage est lui-même divisé en deux grandes parties.

La première traite du langage HTML 5 en tant qu'outil de structuration des documents. Elle vous permet d'acquérir une bonne connaissance de tous les éléments disponibles dans ce but. À ce stade, et même si nous indiquons la présentation par défaut liée à chacun d'eux, vous ne devriez pas vous préoccuper outre mesure de cet aspect et ne jamais choisir tel élément en fonction de son aspect prédéterminé, mais en fonction de son rôle logique et sémantique dans la structuration du contenu. C'est cette différence fondamentale de conception initiée par XHTML qui est poursuivie heureusement par HTML 5.

Cette première partie comprend les chapitres suivants.

- Le chapitre 1 est une introduction générale qui présente les notions à connaître pour utiliser un langage de balisage tel HTML 5. Il définit les règles pour créer un document conforme aux spécifications du WHATWG *(Web Hypertext Application Technology Working Group)* qui est à l'origine de HTML 5. Il indique également les outils et les procédures à suivre pour mettre en ligne les pages web que vous allez créer.
- Le chapitre 2 vous aide à créer la structure générale d'une page conforme à HTML 5. Cette structure fait apparaître les éléments essentiels qui sont communs à toutes les pages, comme la déclaration `DOCTYPE`. Vous y trouverez tous ceux qui constituent l'entête d'une page et qui, s'ils ne créent généralement pas de parties visibles dans un navigateur, ont un rôle important souvent négligé. En particulier, ils permettent de lier la page à des ressources externes comme une feuille de style ou des scripts JavaScript.

Certains autres éléments jouent aussi un rôle essentiel dans le référencement de votre site, en incluant des informations sur le document, dites méta-informations. Vous trouverez enfin dans ce chapitre une présentation de l'ensemble des catégories d'éléments incorporables dans le corps d'un document et les types d'inclusions autorisées entre éléments.

- Le chapitre 3 fait tout d'abord un tour d'horizon complet de tous les éléments utiles à la structuration en grandes divisions d'une page. Il aborde ensuite les divisions secondaires en particulier celles qui permettent d'organiser un texte et d'en marquer certains passages. Enfin, nous présentons comment structurer l'information au moyen des différentes formes de listes utilisées couramment dans la création de menus par exemple.
- Le chapitre 4 vous fait découvrir comment insérer les différents types d'éléments multimédias disponibles. Il peut s'agir aussi bien d'images que de musiques ou de vidéos à l'aide des nouveaux éléments audio et vidéo de HTML 5. Nous verrons également comment rendre une image réactive aux clics sur plusieurs zones différentes. Nous aborderons enfin la création de jauge et de barre de progression qui sont aussi des nouveautés HTML 5.
- Le chapitre 5 présente ce qui est la base des documents hypertextes, à savoir la création de liens déclenchés à partir d'un texte, d'un bouton ou d'une image, soit entre les différents éléments d'une même page, soit entre les pages d'un même site pour créer un système de navigation complet. Ces liens peuvent servir également à déclencher le téléchargement de documents externes non affichables dans une page web, à envoyer un e-mail ou encore à lancer un script JavaScript.
- Le chapitre 6 vous propose d'utiliser les différents éléments qui interviennent dans la création de tableaux. Il aborde et définit tout d'abord la structure générale commune à tous les tableaux. Les éléments de création de lignes, de cellules et leurs regroupements sémantiques éventuels, en groupes de ligne ou de colonnes, sont ensuite traités. Les tableaux sont envisagés ici aussi bien pour la structuration de données mais également comme moyen d'organisation d'une page. Nous précisons cependant les limites des tableaux en tant que technique de mise en page, l'usage des styles CSS étant une bonne alternative à ce type d'organisation.
- Le chapitre 7 présente le seul moyen de rendre une page web interactive, en y incorporant des formulaires. Après avoir défini la structure globale de ces derniers, nous décrivons l'ensemble des nombreux nouveaux composants qui permettent de saisir du texte, des mots de passe, des adresses e-mails, des URL, des dates, des nombres, et d'effectuer des choix à l'aide de boutons radio, de cases à cocher ou de listes de sélection d'options. Nous exposons enfin comment réaliser le transfert de fichiers du poste client vers un serveur. Nous montrons également dans ce chapitre comment utiliser des tableaux pour améliorer la structure d'un formulaire.

La seconde partie de cet ouvrage traite de la création des feuilles de styles au moyen de CSS (*Cascading Style Sheets* ou feuilles de styles en cascade). Elles sont le complément indispensable de HTML 5, la séparation du contenu et de la présentation d'un site ayant été bien mise en évidence dans la première partie.

- Dans le chapitre 8, vous découvrirez le fonctionnement des CSS et la syntaxe utilisée dans la création des styles applicables à un document HTML. Ce chapitre constitue une étape essentielle dans l'apprentissage des CSS car il aborde les nombreux sélecteurs CSS 3 qui permettent, entre autre, d'appliquer un même style à toutes les occurrences d'un élément ou d'appliquer des styles différents à un même élément en fonction de son contexte. C'est du bon usage de ces sélecteurs que dépendra toute la puissance et la diversité d'utilisation des styles que vous allez créer par la suite. Nous y étudions enfin les règles d'héritage et de cascade des propriétés CSS.
- Le chapitre 9 aborde les propriétés de gestion des couleurs, aussi bien pour le texte que pour le fond d'un élément, puis celles destinées à la création des images de fond et leurs différents types de positionnement dans tous les éléments HTML. S'ajoutent à cela les différentes améliorations apportées par CSS 3 pour obtenir des images de fond multiples ou des dégradés de couleurs.
- Le chapitre 10 présente tout d'abord le modèle de boîte applicable à tous les éléments CSS, puis traite de la création des bordures (style, épaisseur, couleur, image) applicables à chaque élément individuellement. Nous y abordons également la création des marges entre la boîte d'un élément et son environnement, ainsi que l'espacement entre son contenu et ses bordures. Toutes les propriétés exposées ici sont de nature à affiner la présentation à l'intérieur d'un document. La notion de contour y est également étudiée. Par rapport à la version précédente, CSS 3 propose des bordures multicolores ou contenant des images, des angles arrondis sans avoir recours à des images, comme c'était le cas auparavant, et des ombres très faciles à créer.
- Le chapitre 11 dresse un panorama des propriétés applicables aux textes, qu'il s'agisse du choix d'une police de caractères, des différentes façons de définir sa taille de manière absolue ou relative par rapport au contexte, du choix de sa couleur, de son style, de sa casse, de sa graisse, sans oublier les nombreuses autres possibilités décoratives comme l'ombrage de texte. Nous décrivons aussi la gestion des interlignes, de l'alignement, de la césure, de l'indentation et de l'espacement du texte. Ce chapitre présente enfin les propriétés spécifiques aux liens hypertextes et les sélecteurs spécifiques aux effets dynamiques.
- Le chapitre 12 apporte les éléments essentiels à la présentation et à la mise en page globale d'un document. Nous y étudions les méthodes de dimensionnement des éléments ainsi que les méthodes de positionnement qui sont des avancées essentielles de l'association CSS/HTML. Elles remplacent les méthodes de mise en page habituelles comme celles qui usent et abusent des tableaux ou encore celles qui emploient des cadres. La richesse de ces propriétés permet également d'agir sur la visibilité et l'ordre d'empilement des éléments. Toutes ces propriétés rendent possibles la création des mises en page les plus diverses. De nouvelles propriétés CSS 3 vous donnent aussi la faculté de présenter du texte sur plusieurs colonnes de façon très simple et de laisser l'utilisateur redimensionner un élément. Enfin, dans le domaine des effets visuels, CSS 3 s'est enrichi de propriétés pour transformer les éléments, qu'il s'agisse de translation, de rotation, de changement de taille ou de déformations diverses, tout ceci

pouvant être déclenché par le visiteur de la page. À cela s'ajoute encore l'opportunité de gérer ces modifications afin de créer des animations dynamiques.

- Le chapitre 13 est spécialement dédié aux tableaux qui possèdent un modèle de gestion particulier. Nous montrons comment traiter la couleur des cellules en fonction, par exemple, de leur appartenance à un groupe de lignes ou de colonnes. Nous étudions également la gestion des bordures des cellules, de la détermination de la largeur des colonnes ou de l'ensemble d'un tableau, des alignements spécifiques à l'intérieur des cellules ou des groupes, etc.
- Le chapitre 14 est destiné spécifiquement aux listes et aux menus qui constituent un moyen de structuration efficace, très approprié aux menus par exemple. Nous y abordons les multiples styles de numérotation disponibles pour les listes ordonnées ou à puces graphiques et leur position par rapport aux items. L'emploi des compteurs est une autre façon de numéroter automatiquement des listes générées dynamiquement. La modification du rendu habituel des éléments nous permet de créer des listes en ligne, constituant par exemple des menus horizontaux, ou encore le rendu sous forme de liste d'un ensemble d'éléments dont ce n'est pas la vocation initiale.
- Le chapitre 15 présente plusieurs compléments utiles qui n'ont pas trouvé leur place ailleurs ! Il s'agit d'abord des Media Queries qui constituent l'outil d'adaptation automatique des styles d'une page en fonction du terminal qui la lit, et en particulier en fonction de la taille de son écran qui peut être très variée. Cette adaptation est aussi consacrée aux moyens disponibles pour obtenir un rendu correct du contenu d'un document web à l'impression, et plus généralement sur tout support constitué de pages calibrées (fichiers PDF, présentations de style diaporama…).

  Nous abordons ensuite le stockage local pour enregistrer des données, y compris en grande quantité, sur le poste client bien mieux que ne le font les cookies. À titre d'exemple nous y développons une application de gestion de contacts. Enfin, nous donnons une initiation à l'incorporation de dessins vectoriels SVG *(Scalable Vector Graphics)* dans une page HTML 5 pour vous mettre l'eau à la bouche sur cette technologie très riche de possibilités et encore pas assez connue. Sont abordées ici la création de formes géométriques, aussi bien que les styles que l'on peut leur appliquer, et la manipulation du texte vectoriel. D'autres fonctionnalités complémentaires comme les animations SVG seront présentées sur mon site (www.funhtml.com) dans plusieurs nouveaux exemples.
- Enfin, les annexes A, B, C et D proposent des références sur des éléments HTML et de leurs attributs, des propriétés CSS 3 (essentiellement celles qui fonctionnent aujourd'hui) et de leurs valeurs, des codes de couleurs conseillées sur le Web et des entités de caractères.

Les exercices proposés à la fin de chaque chapitre vous proposent de mettre en œuvre immédiatement les points étudiés et de tester l'ensemble des connaissances acquises.

Les corrigés de ces exercices ne figurent pas dans cet ouvrage pour ne pas l'alourdir inutilement, mais ils sont téléchargeables librement sur le site des éditions Eyrolles (www.editions-eyrolles.com) et sur mon site dédié à ce sujet (www.funhtml.com). Ils vous permettront de mesurer votre compréhension des notions abordées.

Pour être tout à fait honnête, je me dois de signaler aussi ce que vous ne trouverez pas dans cet ouvrage et qui existe pourtant dans HTML 5.

- L'élément `<canvas>` qui fait beaucoup de bruit chez certains et qui est une sorte d'application de dessin bitmap dans un navigateur mais qui demande beaucoup de code JavaScript, ce qui n'était pas supposé être un acquis pour les lecteurs.
- Les bases de données SQL Web qui sont un sujet très intéressant qui demandent également des connaissances en JavaScript et SQL que nous n'aurions pas eu la place de développer ici.
- Le drag and drop (glisser-déposer).

Notez que la géolocalisation ne figure pas non plus dans ce livre ; ce n'est d'ailleurs pas du HTML 5.

Sur ces sujets on pourra se référer utilement à d'autres ouvrages aux éditions Eyrolles.

Enfin, vous trouverez aussi sur le site www.funhtml.com au fur et à mesure des nouvelles possibilités des navigateurs, des bonus sur certains points non abordés ici.

1

Introduction à HTML 5

Après XHTML 1.1 on attendait, et moi le premier, une version XHTML 2.0 pour confirmer l'orientation prise par le W3C *(World Wide Web Consortium)*, l'organisme qui édite les recommandations des langages du Web. Or, le W3C n'est pas à l'origine de HTML 5. Évolution positive ou régression ? Contrairement aux avis qui courent sur le Web, je dirais qu'il s'agit des deux à la fois, mais aussi que ce langage peut constituer un progrès ; c'est l'objet de ce livre.

Progrès parce que sans se réclamer le moins du monde de XML et d'une rigueur qui était celle de XHTML, il introduit nombre d'éléments qui accentuent son rôle de langage de balisage sémantique. Sémantique parce qu'il n'est plus question d'utiliser le même élément pour des contenus aux sens très différents, mais au contraire d'employer des balises dont le nom indique clairement le contenu. Ainsi, `<header>` est évidemment un en-tête, pas nécessairement celui d'une page, mais toujours un en-tête et pas un pied de page. Il faut reconnaître que cette démarche est la bonne et qu'elle convient parfaitement aux moteurs de recherche dont la tâche d'indexation s'en trouve facilitée.

Régression parce que se trouvent réintroduites des pratiques qui ne devraient pas à mon sens être tolérées par un informaticien. On sait que dans le code d'un programme l'oubli d'un point-virgule en fin de ligne peut bloquer son exécution. L'informatique a des bases mathématiques et réclame donc des habitudes rigoureuses. Un cercle n'est pas un disque : c'est une différence de nature même ! J'ai donc préféré ne pas indiquer toutes les tolérances qu'accepte HTML 5 dans l'écriture du code par rapport aux exigences de XHTML, pour que vous ne les adoptiez pas, dans votre intérêt d'ailleurs. Le laxisme laisse toujours des traces mais c'est peut être encore une idée à contre-courant. En travaillant avec une syntaxe rigoureuse, vous obtiendrez du code propre et qui sera sans doute analysé et affiché plus vite par les navigateurs.

J'ai donc adopté dans ce livre le parti pris de profiter des progrès qu'apporte HTML 5 tout en gardant les bonnes pratiques de formation du code initiées par HTML 4 strict puis XHTML. Le code résultant est évidemment conforme aux spécifications HTML 5 mais, en plus, il est bien écrit.

Généalogie de HTML 5

HTML 5 *(HyperText Markup Language)* est un langage de balisage (dit aussi langage de marquage) qui permet de structurer le contenu des pages web dans différents éléments. Voilà une définition bien abstraite, reconnaissons-le, mais nous y reviendrons en détail dans la section suivante en présentant la notion de balisage.

Historiquement, les langages de balisage sont issus du langage SGML *(Standard Generalized Markup Language)* créé en 1986 pour structurer des contenus très divers. Ce langage s'est révélé trop complexe pour être appliqué tel quel au Web, d'où la nécessité d'en créer une version allégée respectant les mêmes principes essentiels.

L'inventeur du HTML (1992), Tim Berners-Lee, l'avait conçu à l'origine comme un outil de structuration des contenus, principalement textuels, et non pas pour créer des présentations diversifiées. Ce sont les développements successifs, l'essor populaire du Web et la concurrence acharnée entre Netscape et Microsoft pour s'emparer du marché des navigateurs, qui ont détourné HTML de sa vocation première avec l'ajout d'éléments de présentation qui n'avaient rien à y faire. Voulant faire mieux que l'autre, chacun des deux grands a empilé des couches superflues sur HTML. Il faut également reconnaître que l'entrée du Web dans le grand public nécessitait de répondre à une demande d'interfaces graphiques plus esthétiques.

L'absence d'un langage particulier dédié uniquement à la présentation poussait effectivement les webmestres à utiliser tous les moyens pour créer des présentations visuelles agréables. L'apparition de CSS *(Cascading Styles Sheets),* créé en 1996 par Håkon Wium Lie, aurait dû résoudre le problème du détournement de HTML de sa destination première. Mais les mauvaises habitudes étaient prises et la facilité faisait le reste.

L'apparition de HTML 4, et particulièrement de sa version *strict* associée à l'emploi systématique de CSS 2 (publié en 1998), pouvait apporter une solution efficace à ce problème. La création de XML *(eXtensible Markup Language)* en 1998 et son succès dans de multiples domaines d'application ont conduit le W3C à créer le langage XHTML, non plus comme une nouvelle version de HTML, mais comme une reformulation de HTML en tant qu'application XML. Au niveau des éléments et des attributs disponibles, il existait à vrai dire très peu de différences entre HTML 4 strict et XHTML 1.1.

L'impossibilité pour le W3C de trouver un consensus entre les éditeurs de navigateurs et les créateurs de moteurs de recherche pour faire évoluer XHTML a conduit un groupe indépendant, le WHATWG *(Web Hypertext Application Technology Working Group)* dirigé par Ian Hickson (aujourd'hui chez Google !), à entamer le développement de HTML 5.

Les éléments, balises et attributs

Mais au juste comment fonctionne HTML 5 et qu'est-ce qu'un langage de balisage ?

Vous avez sûrement déjà utilisé un traitement de texte tel que Word. Votre texte peut comprendre des titres, des paragraphes, des images, des tableaux, et vous pouvez employer

différentes polices et tailles de caractères dans le même document. Le document final que vous avez réalisé ne laisse apparaître que le résultat de votre mise en page, mais en arrière-plan, votre traitement de texte a enregistré tous les paramètres de mise en page que vous avez utilisés en plus du texte lui-même.

Dans un langage de balisage, tout contenu, qu'il s'agisse de texte, d'images ou d'éléments multimédias les plus divers, doit être enfermé dans un élément. En HTML, chaque élément possède un nom déterminé ; la liste des éléments utilisables est limitative et clairement définie dans la spécification du langage. C'est la grande différence entre HTML et XML, langage dans lequel c'est le programmeur qui crée ses propres éléments selon ses besoins. À quelques exceptions près, un élément a la structure suivante :

```
<nom_element> Contenu </nom_element>
```

Son contenu est précédé par une balise d'ouverture `<nom_element>` et suivi par une balise de fermeture `</nom_element>`. Toutes les balises d'ouverture (ou marqueur) commencent par le signe < et se terminent par le signe >. La balise de fermeture suit la même règle mais le nom de l'élément est précédé d'un *slash* (/). Les navigateurs interprètent donc les contenus en fonction du nom de l'élément et attribuent un style par défaut à chacun de ses contenus.

Les caractéristiques de chaque élément peuvent être précisées par des informations complémentaires que l'on désigne en tant qu'attributs de l'élément. Il peut s'agir par exemple de la définition de la largeur, de la hauteur ou de l'adresse du contenu. Comme nous le verrons dans les sections suivantes, un certain nombre d'attributs sont communs à quasiment tous les éléments de base.

Les attributs d'un élément sont toujours définis dans la balise d'ouverture et doivent être séparés les uns des autres par au moins un espace typographique. Chaque attribut a généralement une valeur, même s'il ne peut prendre qu'une valeur unique. Presque toutes les valeurs ne sont donc pas implicites du moment que l'attribut figure dans la balise d'ouverture. La présence de certains attributs est obligatoire dans quelques éléments particuliers, ce que nous préciserons systématiquement le cas échéant. La plupart du temps, les attributs d'un élément sont facultatifs et il appartient au programmeur de déterminer leur définition par rapport au cas qu'il doit traiter. Nombre d'attributs ont une valeur par défaut. Cela signifie que même si on ne les définit pas dans l'élément, celui-ci se comporte comme si c'était le cas. Il est donc important de connaître ce type d'attribut et de ne pas négliger de les définir avec une autre valeur si ce comportement par défaut n'est pas désiré. La valeur de tous les attributs doit être définie entre guillemets. La syntaxe conforme d'un élément ayant des attributs est donc la suivante :

```
<nom_element attribut1="valeur1" attribut2="valeur2" > Contenu de l'élément
➥ </nom_element>
```

Le contenu d'un élément peut être constitué de texte ou d'autres éléments qui, eux-mêmes, peuvent en contenir d'autres, et ainsi de suite. Cet ensemble d'inclusion constitue la hiérarchie du document HTML 5.

Les attributs de base de HTML

Dans leur quasi-totalité, les éléments disponibles en HTML ont en commun un ensemble d'attributs ayant chacun le même rôle. Ces attributs se répartissent en trois catégories. Chaque élément peut posséder par ailleurs d'autres attributs particuliers. Quand nous définirons par la suite les différents éléments, nous signalerons s'ils possèdent ces attributs sans rappeler leur définition.

Les attributs globaux (ou noyau)

Ils s'appliquent à quasiment tous les éléments.

- L'attribut `accesskey` définit la touche de raccourci clavier pour accéder à un élément.
- L'attribut `id` sert à identifier un élément de manière unique en lui donnant un nom, soit pour lui attribuer un style, soit pour y faire référence sans ambiguïté dans un script JavaScript.
- L'attribut `class` contient le nom d'une classe CSS qui contient des définitions de styles. Comme nous le verrons dans la seconde partie, son usage est très répandu pour affecter ponctuellement des styles à un élément.
- L'attribut `contenteditable`, qui est un booléen, prend les valeurs `true` ou `false`. Si la valeur est `true` le contenu de l'élément en général textuel est modifiable par le visiteur.
- L'attribut `contextmenu` contient le nom de l'identifiant d'un menu contextuel (créé avec l'élément `<menu>`) qui s'affiche quand on utilise l'élément auquel il est attaché (pas encore opérationnel).
- L'attribut `draggable` prend les valeurs booléennes `true` ou `false` qui indiquent si l'élément est déplaçable dans la page. Peu fonctionnel pour l'instant.
- L'attribut `dropzone` définit les contenus que l'on peut déposer dans l'élément auquel il se rapporte.
- L'attribut `dir` indique le sens de lecture du contenu textuel d'un élément ; il peut prendre les valeurs `ltr` (lecture de gauche à droite) ou `rtl` (de droite à gauche).
- L'attribut `hidden` prend une valeur booléenne `true` ou `false` et indique que l'élément ne sera pas visible dans la page (valeur `true`). Il faudra utiliser un script JavaScript pour modifier sa valeur et le faire apparaître (valeur `false`).
- L'attribut `lang` contient le code de la langue utilisée dans l'élément.
- L'attribut `style` permet de définir un style localement pour un élément donné. Il est encore accepté en HTML mais déconseillé au profit des styles regroupés dans l'élément `<style>` ou dans un fichier externe à l'extension `.css`, ces solutions permettant une maintenance plus facile.
- L'attribut `tabindex` est généralisé à tous les éléments. Sa valeur est un nombre entier qui donne une position à l'élément dans l'ordre des tabulations.
- L'attribut `title` contient un texte qui apparaît dans une bulle quand l'utilisateur positionne le curseur quelques instants (ce n'est pas instantané) sur un élément. Le texte qu'il contient peut servir à fournir une information ou une explication sur le rôle de l'élément.

Les attributs de gestion d'événements

Ces attributs permettent de gérer les événements dont un élément peut être le siège et qui sont créés par l'utilisateur. Leur contenu est un script écrit en général en langage JavaScript. HTML 5 définit un grand nombre d'attributs de gestion d'événements, y compris pour des éléments qui ne peuvent pas être le support de ces événements. Il appartient donc aux programmeurs d'effectuer des tests pour vérifier la réalité des événements pour un élément donné. Vous trouverez ci-après la liste des gestionnaires de base et la description de l'événement correspondant.

Tableau 1-1. Les attributs gestionnaires d'événements communs (suite page 8)

attribut	Action de l'utilisateur
`onabort`	Arrêt d'une opération
`oncanplay`	Un fichier multimédia peut être lu (au moins son début)
`oncanplaythrough`	Un fichier multimédia peut être lu entièrement
`onclick`	Clic sur le contenu de l'élément
`oncontextmenu`	Un menu contextuel est déclenché
`ondblclick`	Double-clic sur le contenu de l'élément
`ondrag`	Un élément est en cours de déplacement
`ondragend`	Le déplacement est fini
`ondragenter`	Un déplacement a une cible qui l'accepte
`ondragleave`	L'origine du déplacement est acceptable
`ondragover`	Un élément déplacé survole l'élément concerné par l'attribut
`ondragstart`	Début du déplacement
`ondrop`	Dépose d'un élément
`ondurationchange`	Modification de l'attribut `duration` d'un élément multimédia
`onemptied`	Un élément multimédia n'est plus utilisable (chargement incomplet ou interrompu)
`onended`	Fin de lecture d'un son ou d'une vidéo
`oninput`	Un élément de formulaire est complété
`oninvalid`	Une saisie est non valable (par exemple en dehors d'un intervalle précisé)
`onkeydown`	Maintien d'une touche enfoncée
`onkeypress`	Frappe sur une touche
`onkeyup`	Relâchement d'une touche enfoncée
`onloadeddata`	Une donnée est chargée (comme un son, une vidéo)
`onloadedmetadata`	Une métadonnée est chargée (comme un script, des styles externes)

attribut	Action de l'utilisateur
onloadstart	Début d'un téléchargement d'un média
onmousedown	Enfoncement d'un bouton de la souris
onmousemove	Le curseur de la souris bouge dans la zone de l'élément
onmouseout	Le curseur de la souris quitte la zone de l'élément
onmouseover	Le curseur de la souris est au-dessus de la zone de l'élément
onmouseup	Relâchement d'un bouton de la souris au-dessus de la zone de l'élément
onpause	Pause dans la diffusion
onplay	Démarrage de la lecture
onplaying	Média prêt pour la lecture
onprogress	Média en cours de chargement
onratechange	Modification de la vitesse de lecture
onscroll	Défilement d'un contenu
onstalled	Erreur pendant le chargement d'un média
onsuspend	Arrêt déclenché avant la fin d'un chargement
ontimeupdate	La position actuelle de lecture est modifiée par l'utilisateur
onvolumechange	Modification du volume sonore
onwaiting	Arrêt de la lecture en attente de la disponibilité de la suite

Intérêt de la sémantique

Il s'agit bien plus selon moi d'un changement de pensée et d'organisation qui doit s'opérer dans la création des pages web.

Une page web créée avec HTML doit être pensée en distinguant deux parties.

- Un contenu, structuré au moyen des éléments HTML (grandes divisions, titres, paragraphes, tableaux, images et liens, etc.). À ce stade, et même s'il en a déjà une idée, le créateur ne doit pas nécessairement avoir une vue définitive de la présentation finale. Il lui faut maîtriser principalement l'organisation des informations à fournir à un utilisateur.
- Une feuille de style CSS, définissant la mise en page de ces éléments en fonction du média qui va opérer le rendu du contenu (polices et tailles de caractères, bordures, marges, couleurs, positionnement dans la page, etc.). Les médias se diversifiant en effet de plus en plus en devenant des éléments portables dotés de petits écrans, le traditionnel écran d'ordinateur n'est plus le principal vecteur d'affichage d'une page web. La séparation du contenu et de la présentation étant réalisée, il est possible d'associer à chaque média une feuille de style adaptée au terminal.

L'utilisation de ces méthodes présente les avantages suivants.

- Une meilleure organisation du contenu.
- Une meilleure qualité du code et une plus grande rapidité d'affichage sur les navigateurs récents (Firefox, Internet Explorer, Opera, Safari, Chrome…).
- Une réduction des coûts de développement et de maintenance des sites web ainsi qu'une réutilisabilité accrue et rapide du code. En effet, en ayant respecté les principes précédents, il est très facile de modifier rapidement toute la présentation d'une page sans toucher au code HTML.

Les spécifications HTML et CSS sont aujourd'hui incontournables pour tous ceux qui veulent concevoir un site web de manière professionnelle, et tous les étudiants en informatique et les professionnels du Web se doivent d'acquérir ou de mettre à jour leurs connaissances sur ces techniques.

Règles de base HTML 5

Un document bien formé

Un document HTML doit respecter certaines règles simples.

- Les éléments et les attributs sont insensibles à la casse. Par exemple, `<body>` et `<BODY>` sont acceptés. Choisissez une casse selon votre goût et conservez-la. Par habitude du XHTML, tout le code de ce livre est en minuscules.
- Les éléments non vides doivent avoir une balise d'ouverture et une balise de fermeture. Cela facilite la tâche des navigateurs. Par exemple, plutôt que d'écrire :

```
<ol>
  <li>Item 1
  <li>Item 2
```

on préférera le code suivant :

```
<ol>
  <li>Item1 </li>
  <li>Item2 </li>
</ol>
```

- Les éléments vides ne comportent qu'une seule balise et il est plus correct de les terminer par les caractères `/>` précédés d'un espace pour marquer la fin de l'élément. Par exemple, plutôt que d'écrire :

```
<img src= "monimage.gif"> <hr> <br>
```

on préférera le code suivant :

```
<img src= "monimage.gif" /> <hr /> <br />
```

- Les éléments ne doivent pas se chevaucher et donc obéir au principe premier ouvert, dernier fermé. Dans ce cas, le premier élément est le parent du second et celui-ci est enfant du premier. Par exemple, le code suivant est incorrect :

```
<div> Cette division contient un titre <h1> Important ! </div> </h1>
```

  et doit être remplacé par :

```
<div> Cette division contient un titre <h1> Important ! </h1></div>
```

- Tous les attributs doivent avoir une valeur incluse entre guillemets (`"`). Les différents attributs du même élément doivent être séparés par au moins un espace. Par exemple, plutôt que d'écrire :

```
<p class=styleperso title=attention> Texte important</p>
```

  on préférera le code suivant :

```
<p class="styleperso" title="attention" > Texte important</p>
```

- Une valeur doit être donnée à tous les attributs utilisés, y compris à ceux dont la valeur est unique. Par exemple, plutôt que d'écrire :

```
<input type= "checkbox" checked disabled />
```

  on préférera le code suivant :

```
<input type= "checkbox" checked="checked" disabled="disabled" />
```

  Il y a des exceptions car certains attributs ne doivent pas avoir de valeur en particulier quand elle est booléenne. Dans ce cas, la présence seule du nom de l'attribut signifie implicitement la valeur `true` et l'absence de la valeur `false`.

- Les scripts et les feuilles de style qui contiennent les caractères `<` et `&` peuvent figurer dans des sections CDATA de la façon suivante :

```
<script type="text/javascript">
< ![CDATA[
Code du script…
]]>
</script>
```

- Une autre solution efficace consiste à placer les scripts ou les feuilles de style contenant ces caractères dans des fichiers séparés et à les inclure à l'aide de l'élément `<link>` sur lequel nous reviendrons en détail par la suite.

Un document conforme

Un document HTML 5 se doit également de respecter les règles d'inclusion des éléments les uns dans les autres, telles qu'elles sont définies dans la spécification. En effet, elle définit la liste limitative de tous les éléments HTML 5 utilisables et énumère ceux qui peuvent y être

inclus. Le respect de ces contraintes est impératif pour que le document soit déclaré conforme par le validateur. Vous trouverez à cet effet tout au long de cet ouvrage lors de la description des éléments, et dans l'annexe A, pour chaque élément, la liste de ses éléments enfants (ceux qu'il peut inclure) et de ses éléments parents (ceux dans lesquels il peut être inclus).

Validation d'un document : le label du WHATWG et W3C

Malgré toutes les vérifications auxquelles vous pouvez procéder personnellement, il peut rester des erreurs de conformité dans votre code. Comme peut le faire un compilateur qui signale les éventuelles erreurs de syntaxe, les validateurs du W3C et du WHATWG permettent de vérifier si le code est bien formé et conforme. Pour lancer cette validation automatique, vous devez soumettre l'URL ou le code de vos documents HTML au validateur du W3C accessible à l'adresse suivante : http://validator.w3.org, ou à celui du WHATWG à l'adresse : http://html5.validator.nu/.

La figure 1-1 montre la page de validation du site du whatwg.org dans laquelle plusieurs moyens sont mis à disposition pour valider un document. Dans la liste déroulante (repère ❶) vous pouvez choisir entre saisir l'URL de la page si le document est déjà transféré sur un serveur, choisir le fichier sur votre ordinateur, ou encore copier le code à vérifier directement dans une zone de saisie (repère ❷). Le bouton ❸ entraîne la vérification, et la récompense est l'annonce de la validité du code (repère ❹).

Figure 1-1

La page de validation du WHATWG

L'environnement de travail

Pour créer des pages web avec HTML 5 et CSS 3, il faut être doté d'un environnement de travail adapté. En théorie, un simple éditeur de texte tel que le Bloc-notes de Windows ou Emacs sous Linux peut suffire. Cependant, comme la saisie des noms des différents éléments HTML 5 dans ce type d'éditeur peut devenir à la longue plutôt rébarbative, nous mentionnons quelques outils susceptibles de faire gagner du temps.

Les éditeurs visuels

Dans ce type d'éditeurs, vous travaillez graphiquement sur une page en y incluant des éléments sans saisir une ligne de code. Le plus connu est Dreamweaver, mais il en existe bien d'autres. L'inconvénient de ces éditeurs tient en fait à ce qui paraît être leur avantage : le code que l'on ne saisit pas est généré automatiquement et rien ne garantit qu'il convienne ou qu'il soit conforme aux dernières spécifications. De plus, toute génération automatique éloigne du travail de programmation et ne présente pas un avantage évident pour la création de pages, alors que ce peut être le cas dans un éditeur dédié à un langage de programmation, Java par exemple, et qui peut décharger le programmeur de tâches répétitives.

Outre le temps d'apprentissage et son prix élevé, je ne pense pas qu'il soit utile de faire appel à ce genre d'éditeurs qui auraient tendance à vous rendre passif.

Les éditeurs classiques

Entre le Bloc-notes Windows et Dreamweaver, certains éditeurs offrent un compromis en apportant à la fois une aide à la saisie, qui dispense de taper soi-même le nom des éléments, tout en autorisant de les choisir librement. Outre le fait qu'ils présentent souvent l'avantage d'être gratuits, ces éditeurs, très nombreux et disponibles en téléchargement, permettent de créer rapidement la structure d'un document en utilisant un squelette commun à toutes les pages HTML. Nous donnerons la structure de base d'une telle page au chapitre 2. Il vous restera ensuite à inclure dans le corps de la page les différents éléments qui vont structurer son contenu. Si vous avez procédé à une analyse préalable sur le papier, comme il se doit avant tout codage, cette phase de travail d'inclusion sera très rapide. Parmi les nombreux éditeurs, j'ai choisi pour ma part HTMLPad qui s'avère très pratique pour incorporer rapidement les différents éléments HTML 5 et propriétés CSS 3. Pas tout à fait gratuit mais bon marché, vous pouvez le télécharger y compris en version d'essai complète sur le site : http://www.blumentals.net/htmlpad/.

La figure 1-2 montre l'aspect de l'environnement de travail qu'il fournit. La fenêtre de l'éditeur est divisée en plusieurs zones. La zone ❶ contient la liste des éléments HTML 5 et un double-clic sur l'un d'eux incorpore le code dans la page située au centre à la position du curseur (zone ❷). La zone ❸ contient l'ensemble des propriétés CSS 3 et leurs valeurs quand elles sont à choisir dans une liste ; on peut ainsi incorporer d'un clic n'importe quel style sans erreur de saisie. Cet outil peut faire gagner beaucoup de temps dans la création d'une page.

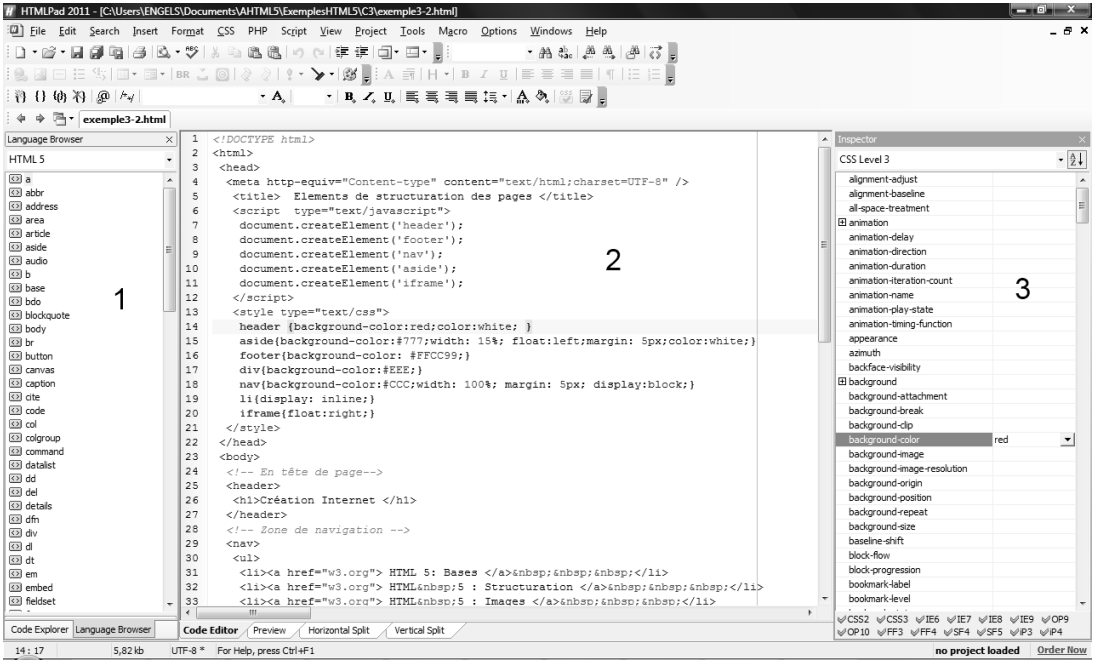


Figure 1-2

L'éditeur HTMLPad

Tests et mise en place du site

Une fois que vous aurez créé l'ensemble des pages de votre site, il faudra vous assurer qu'elles ont un aspect identique ou du moins très semblable dans les différents navigateurs du marché et les résolutions d'écrans afférentes. Ce dernier point doit d'ailleurs avoir fait l'objet d'un choix initial, en particulier si les dimensions retenues pour les différents éléments de la page sont définies de manière fixe (en pixels par exemple), et non en pourcentage, ce qui représente la meilleure solution. Choisir de créer des mises en page en ciblant le format le plus répandu (1 024 par 768 pixels actuellement par exemple) peut produire des résultats désagréables si le poste client est en 800 par 600 pixels ou moins encore.

Si cette phase de test vous amène à constater des résultats divergents et indésirables, c'est sans doute que vous avez utilisé des fonctionnalités non encore prises en compte par un navigateur particulier, et le plus souvent il s'agira de propriétés CSS encore marginales dans Internet Explorer par exemple. Il vous appartiendra alors de renoncer à certaines d'entre elles, au moins provisoirement, pour assurer l'universalité de vos pages si tel est votre objectif. Nous signalons à cet effet dans la seconde partie de cet ouvrage les propriétés qui risquent de poser problème dans certains navigateurs.

Une fois cette phase de test effectuée, il ne reste plus qu'à transférer l'ensemble des pages du site vers le serveur distant qui va les héberger pour les mettre à la disposition de tous. Vous devez pour cela utiliser un logiciel de transfert FTP *(File Transfert Protocol).*

Les hébergements gratuits

Les hébergeurs gratuits de sites personnels peuvent mettre à votre disposition des pages spécialisées permettant ce transfert via une interface web simple. Si vous disposez d'un nom de domaine sur un serveur dédié ou mutualisé, il vous faut généralement utiliser un logiciel FTP.

Il existe de nombreux logiciels de ce type, gratuits ou payants. Je recommande pour ma part d'utiliser FileZilla téléchargeable sur le site http://filezilla.fr/, gratuit et très pratique. La figure 1-3 présente l'interface de FileZilla. Pour établir une connexion avec le serveur et acquérir les droits en vue d'effectuer les transferts de vos fichiers HTML, de vos images ou des supports multimédias, vous devez connaître les paramètres de connexion qui sont fournis par l'hébergeur du site. Il s'agit des données suivantes.

- Hôte : adresse du serveur FTP, par exemple `ftp.funhtml.com` (repère ❶).
- Identifiant : par exemple `funhtml` (repère ❷).
- Mot de passe d'accès au site (repère ❸).
- Éventuellement, indiquez le port de communication (repère ❹) (par défaut, il s'agit de la valeur 21).

Si ces paramètres sont correctement reconnus, vous avez accès à un gestionnaire de fichiers côté serveur similaire à celui que vous pouvez connaître dans Windows ou Linux (repère ❺). Dans la partie gauche, vous trouvez le même type de gestionnaire de fichiers pour sélectionner les fichiers présents sur votre ordinateur (repère ❻). Il vous suffit alors de réaliser une opération de glisser-déposer entre ces deux zones pour que le transfert commence. Sa durée dépend évidemment de la taille du fichier et de la vitesse de votre connexion. Le dossier principal côté serveur est nommé `www`. C'est dans ce dossier que le serveur ira chercher le fichier nommé `index.html` ou `index.htm` en réponse à la requête générale effectuée sur votre site sous la forme `http://www.votresite.com`. C'est dans ce fichier que vous devez créer la page d'accueil du site (ou la page `index.php` si elle contient du code PHP). Dans ce dossier `www`, vous pouvez créer autant de sous-dossiers que vous le désirez. Si vous constituez par exemple un sous-dossier nommé `html`, son contenu principal devra figurer dans un nouveau fichier nommé encore `index.html`, et il sera alors accessible directement par la requête `http://www.votresite.com/html`. Dans les deux cas (répertoire principal ou sous-répertoire), si vous ne créez pas de fichier nommé `index.html`, l'utilisateur devra écrire une requête complète, comprenant le nom du fichier auquel il veut accéder, de la forme `http://www.votresite.com/html/page1.html`.

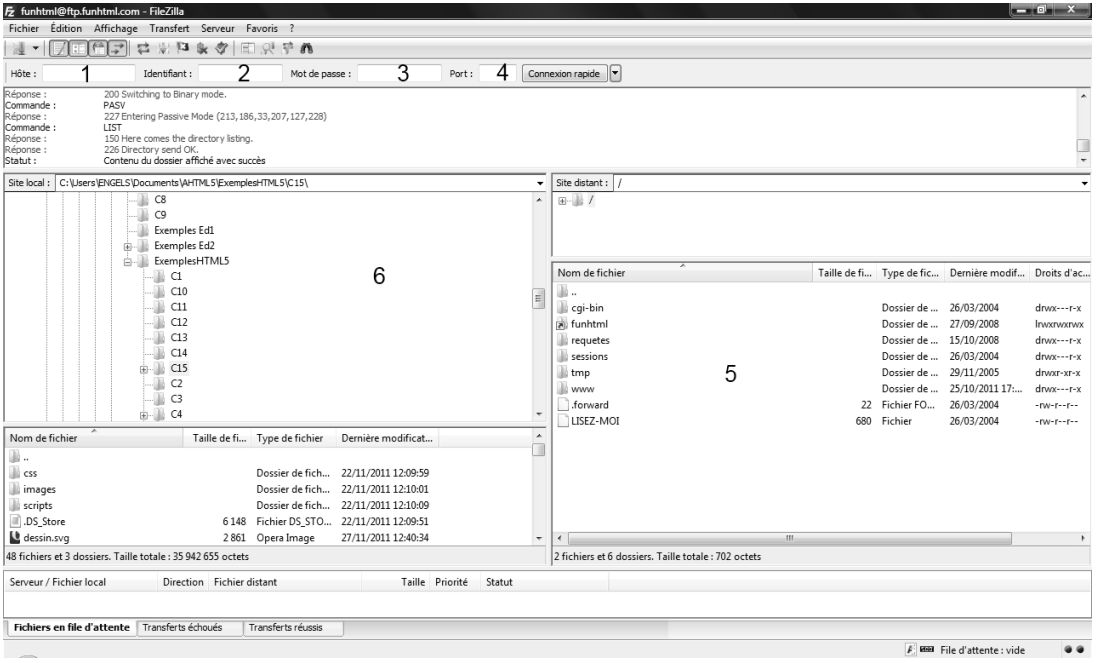


Figure 1-3

L'interface de FileZilla

Référencement du site

Après avoir créé votre superbe site conforme à HTML et CSS, il reste encore une phase importante à accomplir, à savoir le faire connaître. À moins de disposer de moyens publicitaires conséquents, la fréquentation souhaitée ne se produira pas du jour au lendemain. Même si c'est l'adaptation du contenu à un besoin qui fait venir et revenir les internautes, il vous faut d'abord faire connaître l'existence de votre site.

Le moyen le plus simple, et généralement gratuit, est de le référencer dans les annuaires et moteurs de recherche les plus connus comme Google ou Yahoo! afin qu'il soit bien placé en réponse à la recherche d'un internaute. Des ouvrages entiers sont consacrés au référencement, apportant des conseils avisés pour augmenter la fréquentation d'un site. Je recommande le site http://www.abondance.com ou de se procurer l'ouvrage *Réussir son référencement Web* d'Olivier Andrieu aux éditions Eyrolles portant sur le référencement. Nous retiendrons cependant les quelques points suivants.

- Si vous achetez un nom de domaine – ce qui, lié à un hébergement mutualisé, est aujourd'hui possible à bon marché et présente mieux que l'hébergement gratuit offert par votre fournisseur d'accès dont les adresses sont très longues –, choisissez de préférence un nom court et facile à mémoriser. Les noms longs, et particulièrement s'ils sont composés de plusieurs mots, posent soucis à l'utilisateur qui se demande alors si

les mots se suivent ou sont séparés par des tirets, d'où des interprétations divergentes et des erreurs. Certains moteurs de recherche affichent les sites répondant aux mêmes critères par ordre alphabétique et il vaut mieux que votre nom de domaine commence par `a` plutôt que par `z`.

- Les extensions `.com` (ou `.fr` en France) sont préférables à `.org` ou `.net` car ce sont les premières qui viennent à l'esprit des internautes s'ils ont oublié l'extension réelle. La notion de site commercial qui était liée à l'extension `.com` a désormais disparu, et votre site peut présenter aucun aspect commercial et posséder cette extension.
- La définition d'un maximum de mots-clés rapportant objectivement le contenu de votre site est essentielle. Elle doit être réalisée à l'aide de l'élément (voir le chapitre 2) :

```
<meta name="keywords" content="liste des mots clés" />
```

qui est situé dans l'en-tête du document. Cette liste mérite toute votre attention car elle est utilisée par les moteurs de recherche pour indexer vos pages et mettre en adéquation la demande d'un internaute avec les sites qui lui correspondent. Les mots-clés se doivent de refléter fidèlement le contenu du site et les idées qui lui sont associées, et rien d'autre. Inutile par exemple d'identifier les mots les plus recherchés et de les inclure dans votre liste pour attirer du monde. Outre que vous risquez de tromper les internautes, vous pouvez surtout les décevoir si votre contenu ne correspond pas à leur attente. En revanche, n'hésitez pas à fournir une longue liste de mots-clés avec leurs variantes masculin/féminin et singulier/pluriel, ainsi que les mots dérivés qui peuvent correspondre à votre contenu. Vous augmenterez ainsi vos chances de figurer en bonne place dans les résultats des recherches effectuées par les internautes. Les moteurs de recherche se basant aussi sur le contenu des pages, rien ne vous empêche de faire apparaître plusieurs fois dans le contenu les mots importants. Une astuce consistait à répéter ces mots dans la page et à les écrire de la même couleur que le fond de la page en créant des styles CSS appropriés et non plus les balises et les attributs utilisés pour définir les couleurs. Leur répétition est ainsi invisible dans la page, mais perçue par les robots qui analysent le texte.

- Référencez votre site dans tous les principaux moteurs de recherche. Et si vous avez les moyens, Google se fera un plaisir de vous vendre une bonne place dans ses résultats de recherche très objectifs.

2

Structure d'un document HTML 5

Avant de créer des pages web et de leur donner un contenu, nous allons déterminer une structure générale commune à toute page en conformité avec les spécifications HTML 5. En fonction des besoins, les codes des exemples 2-1 à 2-4 serviront de base à la constitution de toutes nos pages. Il suffira donc de les copier dans votre éditeur préféré, puis de les compléter avec un contenu particulier pour chaque page.

Les éléments de base

Le langage HTML 5 est une amélioration du langage HTML 4, avec des simplifications par rapport à la version XHTML qui était de mise avant lui. Tout document peut donc débuter de la même manière par la déclaration suivante (exemple 2-1 repère ❶) :

```
<!DOCTYPE html>
```

Vient ensuite l'élément racine `<html>` (repère ❷) qui inclut les éléments `<head>` (repère ❸) et `<body>` (repère ❻). Chacun de ces éléments a un contenu et donc une balise d'ouverture et une balise de fermeture, `<head>` incluant obligatoirement l'élément `<title>` (repère ❺) et un élément `<meta />` (repère ❹) qui contient la définition du jeu de caractères utilisé dans la page, et `<body>` ayant au moins un élément enfant ; ici, il s'agit de `<h1>` (repère ❼). La structure minimale d'un document HTML 5 est donc semblable à celle de l'exemple 2-1. Le fichier contenant ce code doit avoir une extension `.html` ou `.htm`.

Exemple 2-1 Structure minimale d'un document HTML 5

```
<!DOCTYPE html> ❶
<html> ❷
 <head> ❸
  <meta http-equiv="Content-type" content="text/html;charset=UTF-8" /> ❹
  <title> HTML 5 et CSS 3 </title> ❺
 </head>
 <body> ❻
  <!-- Tout le contenu de la page -->
  <h1>Le corps de la page minimale</h1> ❼
 </body>
</html>
```

Nous retrouvons bien dans cet exemple la structure arborescente décrite au chapitre 1. L'élément racine, au sens XML du terme, est `<html>` et inclut les éléments `<head>` et `<body>`. L'élément `<head>` contient l'élément `<title>` qui est obligatoire ainsi que la déclaration du jeu de caractères dans un élément `<meta />` ; l'élément `<body>`, qui ne doit pas être vide (ce qui est évident), contient un titre de niveau 1 `<h1>` sur lequel nous reviendrons plus loin. Du point de vue hiérarchique, `<html>` est bien le parent ou l'ancêtre de tous les autres.

Les commentaires

Tout ce qui est contenu entre les symboles `<!--` et `-->` est considéré par le navigateur comme des commentaires et n'est pas affiché dans la page, même s'ils se trouvent dans l'élément `<body>`. Comme pour tout langage de programmation, nous avons tout intérêt à commenter le code HTML afin d'en faciliter la relecture a posteriori. Notez toutefois que les commentaires seront visibles par l'internaute si celui-ci choisit d'afficher le code source de la page dans son navigateur. Évitez donc d'y inclure des informations confidentielles et d'y faire figurer des informations privées.

Un document HTML 5 peut incorporer du code PHP pour créer des pages dynamiques (interaction avec une base de données et création automatique de pages). Dans ce cas, le code PHP est compris entre les balises `<?php` et `?>`. Notez que le code de l'exemple 2-2 passé au validateur n'est pas déclaré conforme car il ne reconnaît pas ces balises, mais le code créé par l'exemple sera conforme.

Pour éviter les problèmes d'interprétation divergente entre les différents navigateurs, nous utiliserons systématiquement la déclaration du jeu de caractères avec l'élément `<meta />` dans chaque document. La structure minimale de ce type de page est donc celle de l'exemple 2-2. Notez que le fichier PHP a une extension propre, du type `.php` ou `.php5` par exemple, toujours en fonction de la configuration du serveur. Pour que le document HTML, que le serveur va finalement envoyer au navigateur, soit conforme au standard, il faut que le premier script placé au début du document (repère ❶) ne crée aucun code HTML (il peut par exemple ne contenir que des fonctions) et que le second (repère ❷) ne crée que du code HTML conforme. En respectant ces conditions, il n'y a aucune limite à l'utilisation de scripts PHP à l'intérieur d'un document.

Exemple 2-2. Structure d'une page HTML 5 contenant un script PHP

```
<?php ❶
// Placez ici du code PHP
?>
<!DOCTYPE html>
<html lang="fr">
 <head>
   <meta http-equiv="Content-type" content="text/html;charset=UTF-8" />
   <title> HTML 5 et CSS 3 </title>
  </head>
 <body>
  <!-- Le corps de la page -->
  <h1>Le corps de la page</h1>
  <?php ❷
   echo "<h2> Placez ici du code PHP créant un titre</h2>";
   ?>
 </body>
</html>
```

Dans toute la suite de cet ouvrage, nous n'utiliserons que la structure de base présentée à l'exemple 2-1.

La déclaration DOCTYPE

Nous avons déjà indiqué que le code d'une page HTML devait se conformer à des règles précises. La déclaration `DOCTYPE`, obligatoire dans tout document, précise le type de document qui va être créé. Dans HTML 5, cette déclaration est désormais réduite à sa plus simple expression par rapport à XHTML :

```
<!DOCTYPE html>
```

Dans ce code, la partie `html` donne le nom de l'élément racine du document.

L'élément racine <html>

L'élément `<html>` est l'élément racine du document, au sens XML du terme. C'est donc lui qui est le parent de tous les autres, soit directement, comme `<head>` et `<body>`, soit indirectement par l'intermédiaire de ces derniers. Il est donc le conteneur de premier niveau placé en haut de la hiérarchie de tous les éléments du document. Il n'existe que deux éléments enfants de l'élément `<html>`. En HTML 5, son contenu est constitué de l'en-tête du document, introduit par la balise `<head>` et terminé par la balise `</head>`, puis par le corps du document introduit par `<body>` et terminé par `</body>`, comme nous pouvons le vérifier dans les exemples.

L'élément racine possède les attributs communs dont les plus utiles sont :

- l'attribut `lang` dont la valeur est un code de langue normalisée qui indique la langue utilisée par défaut dans la page. Cette valeur sera reconnue par les moteurs de recherche

pour leur permettre d'indexer les pages du site en effectuant un tri par langue. Elles n'apparaîtront dans Google par exemple que si l'utilisateur a choisi le bouton France ;

- l'attribut `dir` qui indique le sens de lecture du texte de la page. Il peut prendre les valeurs `ltr` pour le texte qui se lit de gauche à droite (langues européennes) ou `rtl` pour le texte qui se lit de droite à gauche (langues orientales : hébreu, arabe).

Un élément `<html>` complet tel que nous pouvons l'utiliser s'écrira donc :

```
<html lang="fr" dir="ltr">
 <!--suite des éléments inclus -->
</html>
```

En pratique, pour des sites ayant un contenu dans une langue européenne, nous omettrons l'attribut `dir` ; dans nos exemples, nous n'utiliserons pas systématiquement l'attribut `lang`.

L'en-tête d'un document : l'élément <head>

L'élément `<head>` englobe un certain nombre d'informations utiles au bon affichage de la page web. Ces informations dites métadonnées sont contenues dans six éléments différents qui ont chacun un rôle bien déterminé. Il s'agit des éléments `<base/>`, `<link />`, `<meta />`, `<script>`, `<style>` et `<title>` dont nous allons étudier les rôles respectifs.

Aucun d'eux n'a de répercussion directement visible dans la page et seul le contenu de l'élément `<title>` sera visible, non dans la page mais dans la zone de titre du navigateur. Le bloc d'en-tête a donc la structure suivante, dans laquelle seuls les éléments `<title>` et `<base />` ne doivent figurer qu'une seule et unique fois, les autres n'ayant pas de limites.

```
<head>
  <title>Titre de la page</title>
  <meta http-equiv="Content-type" content="text/html;charset=UTF-8" />
  <base href="http://www.monsite.com" />
  <link rel="shortcut icon" type="images/x-icon"   href="../images/favicon.ico" />
  <meta name="Author" content="Jean ENGELS" />
  <script  type="text/javascript">
   <!-- Scripts JavaScript -->
  </script>
  <style type="text/css">
   <!-- Styles CSS -->
  </style>
</head>
```

Remarquons d'emblée que seuls les éléments `<title>`, `<script>` et `<style>` ont un contenu, et donc une balise fermante.

Nous allons maintenant détailler le rôle de chacun des éléments inclus dans `<head>`.

L'adresse de base : l'élément <base />

Il s'agit d'un élément vide et n'a donc pas de balise de fermeture. L'information qu'il contient est donnée dans son unique attribut `href` dont l'utilisation est obligatoire. Le

contenu de cet attribut est une URL qui fournit l'adresse de base de tous les fichiers utilisés dans la page quand leur adresse est transmise de manière relative. Si nous écrivons le code suivant :

```
<base href="http://www.funhtml.com/" />
```

le navigateur ira chercher une image dont l'URL est indiquée sur le serveur par `/html/images/monimage.gif` à l'adresse :

```
http://www.funhtml.com/html/images/monimage.gif
```

L'élément `<base />` possède également l'attribut commun `id`, qui ne peut servir qu'à modifier la valeur de l'attribut `href` au moyen d'un script JavaScript, selon la syntaxe suivante :

```
document.getElementById(id_element).href='valeur'
```

Les documents liés : l'élément <link />

Comme le précédent élément, il s'agit d'un élément vide dont l'information est contenue dans ses attributs. Il permet d'établir un lien entre la page HTML 5 en cours et un autre document externe nécessaire à la page. Nous l'utiliserons particulièrement pour lier la page à une feuille de style CSS contenue dans un fichier ayant l'extension `.css` ou un script JavaScript contenu dans un fichier sous l'extension `.js`.

L'utilisation de l'élément `<link />` crée l'incorporation virtuelle de ces documents dans le code de la page web. On parle d'incorporation virtuelle car la page se comportera comme si le code des fichiers externes faisait partie de la page, le contenu de ces fichiers n'étant pas visible, même en affichant le code source de la page.

La liaison avec les fichiers externes est déterminée par les attributs `rel`, `type`, `href`, `hreflang`, `media` et `sizes`.

- L'attribut `rel` indique le nom de la relation à établir avec le fichier externe. Il peut prendre les valeurs suivantes :
  - `rel="stylesheet"` si le fichier externe est une feuille de style.
  - `rel="alternate"` si le fichier est une page alternative (de rechange, proposée aux visiteurs dans les navigateurs).
  - `rel="shortcut icon"` ou `"icon"` pour faire référence à l'icône identifiant le site et qui s'affiche devant l'adresse dans les navigateurs les plus conformes.
  - `rel="previous"` ou `rel="prev"` si le fichier désigné est la page précédente dans l'ordre normal de consultation du site.
  - `rel="next"` si le fichier est la page suivante dans l'ordre normal de consultation du site.
  - `rel="help"` si le fichier est la page d'aide.

- L'attribut `type` précise le type de contenu du fichier externe. Il peut par exemple prendre les valeurs suivantes :

```
type = "text/css" pour une feuille de style CSS
type = "text/javascript" pour un script JavaScript
type = "text/html" ou "text/xml"
type = "images/x-icon" pour créer une icône
```

- L'attribut `href` contient l'adresse relative ou absolue de la ressource externe associée.
- L'attribut `hreflang` – qui, comme `lang`, prend pour valeur un code de langue – précise la langue utilisée dans le document cible.
- L'attribut `media` indique le type de média concerné par le document externe. Nous l'utiliserons en particulier pour lier une feuille de style en précisant le type de média visé par les styles du document CSS. Les valeurs de l'attribut `media` sont `screen` (écran d'ordinateur), `print` (imprimante), `tty` (télétype), `tv` (téléviseur), `projections` (rétro ou vidéoprojecteur), `handheld` (PDA, téléphone), `braille` (terminal braille), `aural` (navigateurs oraux) et `all` (tous les médias).
- L'attribut `sizes` indique la taille des icônes en particulier quand l'attribut `rel` vaut `icon`. Sa valeur doit comporter deux nombres entiers séparés par les caractères `x` ou `X`.

À titre d'exemple, nous pouvons écrire les codes suivants pour l'élément `<link/>` :

- Pour lier une feuille de style :

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

- Pour lier plusieurs feuilles de styles en précisant le média concerné :

```
<link rel="stylesheet" type="text/css" href="styleWeb1.css" media="screen"/>
<link rel="stylesheet" type="text/css" href="styleWeb2.css" media="print"/>
```

- Pour lier un script JavaScript :

```
<link rel="script" type="text/javascript" href="code.js"/>
```

- Pour créer une icône dans la barre d'adresse :

```
<link rel="icon" type="images/x-icon" href="/fashion.icon"/>
```

- Pour créer un lien de dépendance vers un document :

```
<link rel="next" type="text/html" href="page3.html" />
<link rev="prev" type="text/html" href="page1.html" />
```

- Pour créer un lien vers la page d'accueil :

```
<link rel="start" type="text/html" href="index.html" />
```

Les commentaires

Il est toujours utile de commenter votre code HTML 5, comme tout code informatique d'ailleurs, pour en permettre une meilleure compréhension, en particulier pour le relire un certain temps après l'avoir écrit. Tous les commentaires que vous écrirez seront ignorés par le navigateur, et vous pouvez donc vous exprimer librement. Pour placer un commentaire, vous devez l'ouvrir avec les symboles `<!--`, et le fermer avec les symboles `-->`. N'oubliez pas de fermer vos commentaires, sinon tout ce qui suit sera encore interprété en tant que tel, provoquant une erreur. Par exemple, on aura :

```
<!-- Voici un commentaire HTML
qui peut comporter plusieurs lignes
sans problème -->
```

Les méta-informations : l'élément <meta />

L'élément `<meta />` est également un élément vide pour lequel l'information est contenue dans ses attributs. Ces informations ne sont donc pas visibles dans la page mais elles sont destinées au serveur web, aux navigateurs et aux moteurs de recherche. Chaque information est identifiée par un nom et un contenu. Le nom de l'information est défini dans les attributs `name` ou `http-equiv` dont les rôles sont similaires, et la valeur associée est contenue dans l'attribut `content` sous la forme suivante :

```
<meta name="nom1" content="valeur1" />
<meta http-equiv="nom2" content="valeur2" />
```

Si nous utilisons l'attribut `http-equiv`, l'information indiquée dans l'attribut `content` sera présente dans les en-têtes HTTP envoyés par le serveur au navigateur sous la forme de couple nom/valeur.

La plupart des valeurs des attributs `name` et `http-equiv` sont des mots-clés. Nous allons nous arrêter sur la signification et l'utilité des plus courants d'entre eux.

- `name="author"` désigne le nom de l'auteur de la page sans pour autant créer un copyright.

  ```
  <meta name= "author" content="Jean Engels" />
  ```

- `name="keywords"` : cette valeur est très importante pour le créateur de site car son incorporation dans un document sert à l'indexation des pages web par les moteurs de recherche et les annuaires. L'attribut `content` associé à `name` contient la liste des mots-clés que vous allez choisir comme les plus représentatifs du contenu du site. Chaque mot ou expression est séparé des autres par une virgule. Il n'est pas rare d'utiliser plusieurs lignes de mots-clés dans l'attribut `content`. L'utilisation de l'élément `<meta />` à cette fin est donc des plus utile car il va permettre une mise en valeur de votre site qui apparaîtra dans les réponses fournies par les moteurs de recherche si vos mots-clés correspondent à la demande formulée par un internaute. Il est important de bien choisir ses mots-clés pour qu'ils correspondent vraiment au contenu du site et d'en multiplier les variantes dans la liste de l'attribut `content`. On pourra retrouver le même

mot au singulier et au pluriel, au masculin et au féminin. En revanche, il serait contre-productif d'utiliser les mots les plus demandés dans les moteurs de recherche sous prétexte d'attirer du public, alors qu'ils ne correspondent pas au contenu réel de votre site. Exemple de code :

```
<meta name="keywords" content="HTML 4, XHTML, HTML 5, CSS 2, CSS 3, design web,
➥ création de sites />
```

- `name="description"`. Dans le même ordre d'idée que la valeur précédente, il indique une brève description de l'information contenue dans le site. C'est cette description qui apparaît dans le moteur de recherche et il est donc essentiel qu'elle soit courte et correcte. Inutile de donner une description de 10 lignes alors que Google par exemple n'en affiche que deux. Il est également fortement recommandé d'utiliser cet élément `<meta />` car si vous ne fournissez pas vous-même une description de la page, Google et les autres moteurs de recherche utilisent les premières lignes de votre page qui ne sont pas nécessairement les plus explicites. Exemple de code :

```
<meta name="Description" content="Le site du livre &raquo; HTML 5 et CSS 3
➥ &raquo; de Jean Engels Editions Eyrolles" />
```

- `http-equiv="refresh"`. Quand il est associé à l'attribut `content` qui a pour valeur un nombre de `N` secondes, son utilisation permet de forcer le navigateur à recharger la page toutes les `N` secondes. On procédera ainsi pour un site aux informations renouvelées très fréquemment, par exemple un site de cotation boursière, ou pour afficher l'heure régulièrement par exemple à l'aide d'un script JavaScript ou PHP. Vous pouvez également utiliser cet élément pour rediriger automatiquement le visiteur vers une autre page du même site ou encore d'un autre site. On appliquera cette technique si le contenu du site a changé de nom de domaine. Pour cela, l'attribut `content` doit contenir le nombre `N` de secondes avant lequel la redirection sera effectuée, suivi d'un point-virgule (;) et de l'URL absolue de la nouvelle page. Par exemple, pour rediriger vers une autre page au bout de dix secondes, écrivez le code suivant :

```
<meta http-equiv="refresh" content="10; http:/www.funhtml.com/index/>
```

- `http-equiv="Content-type"`. Cette valeur de l'attribut permet de définir simultanément le type du document et le jeu de caractères employé dans la page. Comme nous l'avons déjà signalé, il faut l'utiliser impérativement si le jeu de caractères n'a pas été précisé, ou si cette information est absente, comme dans le cas des pages PHP. Si cette déclaration n'est pas effectuée, l'utilisation de cet élément `<meta />` est indispensable sous peine de voir le document non validé. L'attribut `content` contient alors le type du document suivi d'un point-virgule puis le code du jeu de caractères. Ce qui donne par exemple le code suivant :

```
<meta http-equiv="Content-type" content="text/html;charset=UTF-8" />
```

Les scripts internes : l'élément <script>

Nous avons vu qu'il était possible de lier la page HTML 5 avec un fichier externe contenant du code JavaScript au moyen de l'élément `<link />`. Si cette solution correspond bien au concept de séparation des fichiers ayant chacun un rôle différent, il est également possible de réaliser la même opération avec l'élément `<script>` qui sera alors vide. Le type de langage utilisé est précisé dans l'attribut `type` qui est obligatoire et contient généralement la valeur `text/javascript` ou `application/javascript` pour JavaScript. Dans le cas d'un fichier externe, il faut employer l'attribut `src` qui donne l'URL du fichier externe du script qui possède l'extension `.js` pour JavaScript. Nous aurions par exemple le code suivant :

```
<script type="text/javascript" src="http://www.funhtml/html/fichiercode.js">
</script>
```

Il est toujours possible d'incorporer du code JavaScript dans une page de code au moyen de l'élément `<script>` mais, à la différence de l'utilisation précédente, le code du script est inclus entre les balises `<script>` et `</script>`. Dans ce cas, seul l'attribut `type` est requis. Il est d'usage courant d'inclure tout le code JavaScript dans un commentaire situé à l'intérieur de l'élément `<script>` pour qu'il ne soit pas interprété par les navigateurs dépourvus d'interpréteur (s'il en existait encore !) mais aussi et surtout si le visiteur a volontairement interdit l'exécution des scripts JavaScript.

Nous pouvons par exemple écrire le code suivant :

```
<script  type="text/javascript">
  <!--
  function debut()
  {alert('Bonjour HTML 5');}
  -->
</script>
```

En plus de l'attribut `id` commun à la plupart des éléments, l'élément `<script>` possède également les attributs suivants dont le rôle est annexe par rapport aux précédents.

- `charset` pour préciser le jeu de caractères utilisés dans l'élément.
- `defer` dont la seule valeur autorisée est `defer`. S'il est utilisé, l'interpréteur JavaScript du navigateur n'interprète pas le code contenu dans l'élément avant l'affichage de la page, ce qui rend l'affichage plus rapide. Cet attribut ne sera utilisé que si le code ne contient pas d'instructions provoquant un affichage direct dans la page du type de celles réalisées au moyen de la fonction `write` ().

À la différence des autres éléments présents dans l'en-tête du document, l'élément `<script>` peut être utilisé également dans le corps du document, directement inclut entre les balises `<body>` et `</body>`, ou indirectement dans de nombreux autres éléments inclus eux-mêmes dans `<body>` et dont la liste figure dans le tableau 2-1.

Tableau 2-1

Tous les éléments de la catégorie Phrasing (voir le tableau 2-3) ainsi que `<head>`.

Le code suivant utilise l'élément `<script>` et tous ses attributs :

```
<script type="text/javascript" charset="UTF-8" defer="defer">
<!-- Code JavaScript -- >
</script>
```

L'incorporation des styles : l'élément <style>

L'utilisation des styles CSS est étroitement liée à HTML 5 comme elle l'était déjà avec XHTML. Nous avons vu qu'ils pouvaient le plus souvent être écrits dans un fichier externe lié à la page avec l'élément `<link />`. Cependant, comme pour les scripts, il est possible de les placer directement dans le code HTML en tant que contenu de l'élément `<style>`. Il n'est pas interdit d'écrire tous les styles dans cet élément mais cette possibilité sera réservée aux cas où les styles sont peu nombreux. Il est aussi envisageable de lier un fichier externe contenant les styles communs à toutes les pages d'un site avec `<link />` et d'ajouter des styles particuliers à une page dans l'élément `<style>` de celle-ci.

En plus des attributs `id`, `lang`, `dir` et `title` que nous avons déjà rencontrés, et peu utiles ici, l'élément `<style>` possède les attributs suivants.

- L'attribut `type`, dont la présence est obligatoire, précise le type de feuilles de styles utilisées. Pour les documents HTML, il prend toujours la valeur `text/css`.
- L'attribut `media` précise le type de média concerné par la feuille de style. Il est donc possible d'incorporer plusieurs éléments `<style>` dans l'en-tête de la page, chacun étant destiné à un média différent (voir l'exemple 2-4). Il prend les valeurs suivantes : `screen` (écran d'ordinateur), `print` (imprimante), `tty` (télétype), `tv` (téléviseur), `projections` (rétro ou vidéoprojecteur), `handheld` (PDA, téléphone), `braille` (terminal braille), `aural` (navigateur oral) et `all` (tous les médias).

Le code suivant définit une couleur de fond jaune pour la page et une couleur bleue pour le texte pour le media écran :

```
<style type="text/css" media="screen">
 body{background-color:yellow;color:blue;}
</style>
```

Nous reviendrons en détail sur l'écriture des styles dans la seconde partie de cet ouvrage. Il nous faut simplement retenir ici la localisation des styles.

Le titre de la page : l'élément <title>

Chacun a pu remarquer dans son navigateur qu'avant l'affichage complet d'une page web, un titre apparaît dans la barre de titre située en haut de la fenêtre du navigateur. Ce texte est défini dans l'élément `<title>` qui est l'un des deux seuls dont la présence est obligatoire

dans l'élément <head>. Son contenu est un simple texte qui doit résumer le contenu de la page en une ligne maximum. Il est important de bien réfléchir à ce contenu car c'est aussi lui qui apparaîtra comme titre principal du site dans les moteurs de recherche. Il doit donc être accrocheur et bien correspondre à l'esprit de la page. Nous aurons par exemple le code suivant :

```
<title>Le site de HTML 5 et CSS 3 </title>
```

L'élément <title> possède les attributs globaux dont lang, dir et id que nous avons déjà rencontrés.

L'exemple 2-4 crée une page en utilisant la plupart des éléments possibles de l'en-tête <head> dont <title> (repère ❶), <base /> (repère ❷), un grand nombre d'éléments <meta /> (repère ❸) et <link /> (repère ❹), les éléments <style> (repère ❺) et <script> (repère ❻). L'élément <body> (repère ❼) contient un minimum d'éléments pour obtenir un affichage, à savoir un titre (repère ❽) et une image (repère ❾). Nous reviendrons dans la section suivante sur cet élément important et sur ce qu'il peut contenir.

Exemple 2-4 Les éléments de l'en-tête du document

```
<!DOCTYPE html>
<html>
 <head>
  <meta http-equiv="Content-type" content="text/html;charset=UTF-8" />
    <!-- Element <title> --> ❶
    <title> HTML 5 et CSS 3 </title>
    <!-- Element  <base /> --> ❷
    <base href="http://www.funhtml.com/html/" />
    <!--Elements <meta />  --> ❸
    <meta name="Author" content="Jean ENGELS"/>
    <meta name="Keywords" content="HTML 5, CSS 3, Web" />
    <meta name="Description" content="Le site du livre &raquo; HTML 5 et CSS 3
    ➥ &raquo; de Jean Engels Editions Eyrolles" />
    <meta http-equiv="default-style" content="text/css" />
    <meta http-equiv="Refresh" content="1250; URL=http://www.funhtml.com/xhtml">
    <!-- Elements <link /> --> ❹
    <link rel="shortcut icon" type="images/x-icon" href="/html/images/
    ➥ favicon.ico" />
    <link rel="stylesheet" type="text/css" href="/xhtml/C2/messtyles.css" />
    <link rel="next" title="PHP 5" type="text/html" href="http://www.funhtml.com/
    ➥ php5" />
    <link rel="previous" title="Document HTML" type="text/html" href="/html/C2/
    ➥ exemple2-1.xml" />
    <link rel=alternate href="/en/html" hreflang=en type=text/html title="English
    ➥ HTML">
    <!-- Element <style> --> ❺
    <style type="text/css" media="screen">
      body {background-color:yellow;color:blue;font-size:40px}
    </style>
    <style type="text/css" media="print">
```

```
      body {background-color:white;color:green;}
    </style>
    <!-- Element <script> --> ❻
    <script  type="text/javascript">
     function debut()
     {alert('Bonjour HTML 5');}
    </script>
    <script type="text/javascript" src="http://www.funhtml/html/fichiercode.js">
    </script>
  </head>
  <body onload="debut()"> ❼
    <!-- Tout le contenu de la page -->
   <h1>Le corps de la page</h1> ❽
    <p>
      <img src="/html/images/html5.png" alt="HTML 5" /> ❾
    </p>
  </body>
</html>
```

La figure 2-1 montre le résultat minimal obtenu.

Figure 2-1

L'utilisation des éléments de l'en-tête

Le corps du document : l'élément <body>

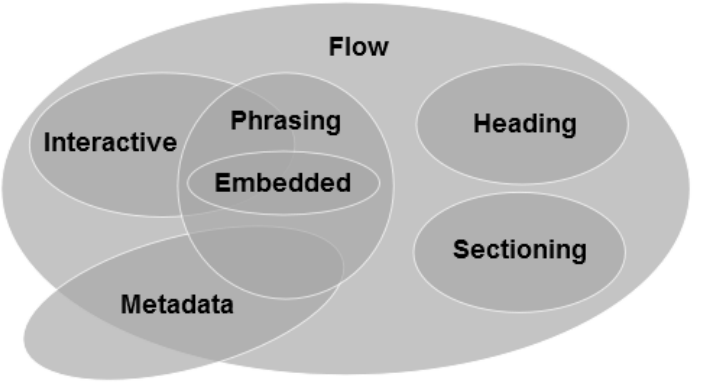
L'essentiel des éléments de l'en-tête `<head>` que nous venons d'aborder ont un contenu invisible dans le navigateur. L'élément `<body>` au contraire est le conteneur de l'ensemble des éléments textuels et graphiques d'une page web.

Les catégories d'éléments

La presque totalité des éléments HTML 5 sont désormais regroupés en catégories qui précisent leur rôle. Celles-ci ne sont pas indépendantes les unes des autres et il existe de nombreuses intersections entre ces différents ensembles comme le montre la figure 2-2.

Figure 2-2

Les catégories d'éléments



Les tableaux 2-2 à 2-8 donnent pour chaque catégorie la liste exhaustive des éléments qui la composent. Le tableau 2-9 donne la liste des éléments HTML 5 ne figurant dans aucune catégorie précise.

Tableau 2-2. L'ensemble des éléments de catégorie Flow

a, abbr, address, article, aside, audio, b, bdo, blockquote, br, button, cite, code, del, dfn, div, dl, em, embed, fieldset, figure, footer, form, h1, h2, h3, h4, h5, h6, header, hgroup, hr, i, iframe, img, input, ins, kbd, label, map, mark, math, menu, meter, nav, noscript, object, ol, output, p, pre, progress, q, s, samp, script, section, select, small, span, strong, sub, sup, svg, table, textarea, u, ul, var, video, wbr, texte brut

Tableau 2-3. L'ensemble des éléments de catégorie Phrasing

abbr, audio, b, bdo, br, button, cite, code, dfn, em, embed, i, iframe, img, input, kbd, label, mark, math, meter, noscript, object, output, progress, q, s, samp, script, select, small, span, strong, sub, sup, svg, textarea, u, var, video, wbr, texte brut

Tableau 2-4. L'ensemble des éléments de catégorie Interactive

a, button, embed, iframe, label, select, textarea, audio (si l'attribut controls est présent), img (si l'attribut usemap est présent), input (sauf pour un composant hidden), object (si l'attribut usemap est présent), video (si l'attribut controls est présent)

Tableau 2-5. L'ensemble des éléments de catégorie Heading

h1, h2, h3, h4, h5, h6, hgroup

Tableau 2-6. L'ensemble des éléments de catégorie Sectioning

article, aside, nav, section

Tableau 2-7. L'ensemble des éléments de catégorie Embedded

audio, embed, iframe, img, object, svg, video

Tableau 2-8. L'ensemble des éléments de catégorie Metadata

base, link, meta, noscript, script, style, title

Tableau 2-9. L'ensemble des éléments hors catégorie

caption, col, colgroup, dd, dt, figcaption, head, html, legend, li, optgroup, option, param, source, tboby, tfoot, th, thead, tr

Au fur et à mesure de leur étude, nous préciserons la catégorie des éléments qui peuvent être inclus dans tel élément. Ceci est utile pour savoir, comme en XHTML, quel peut être le contenu (les enfants) d'un élément et dans quels éléments il peut être inclus (ses parents). Il est important pour être conforme aux prescriptions HTML 5 de respecter les différentes inclusions. C'est donc le premier contrôle auquel vous devez procéder. Par exemple, l'élément `<body>` peut contenir tous les éléments de la catégorie Flow mais l'élément `<caption>` ne peut avoir comme parent que l'élément `<table>` et rien d'autre. Notons malgré tout que même si la catégorie Flow contient le texte brut, il est déconseillé d'inclure directement du texte dans l'élément `<body>` ; il vaut mieux le placer dans un élément enfant de `<body>`, ne serait-ce que pour lui appliquer simplement un style particulier et aussi pour qu'il apparaisse clairement dans la hiérarchie des éléments.

Exemple 2-5 Exemple d'inclusion des éléments dans une page HTML 5

```
<!DOCTYPE html>
<html>
 <head>
  <meta http-equiv="Content-type" content="text/html;charset=UTF-8" />
  <title>Les éléments du corps de la page</title>
  <meta name="Author" content="Jean ENGELS" />
  <meta name="Keywords" content="HTML 5" />
  <meta name="Description" content="Des inclusions" />
  <link rel="shortcut icon" type="images/x-icon" href="../images/favicon.ico" />
  <style type="text/css" title="style1">
       body{background-color:   #EEE;}
  </style>
 </head>
 <body>
  <div>
   <address>Contact : html@funhtml.com</address>
   <blockquote>
    <hgroup>
     <h1>titre de niveau 1</h1>
    </hgroup>
    <dl>
     <dt>Liste de d&eacute;finition</dt>
     <dd>UN</dd>
    </dl>
   </blockquote>
  </div>
  <!--  -->
  <form action="script.php">
   <fieldset>
    <legend>commentaires</legend><br />
    <textarea cols="30" rows="5">texte</textarea>
   </fieldset>
  </form>
  <!--  -->
  <table border="1">
   <tbody>
    <tr>
     <td>Cellule 1 de tableau</td>
    </tr>
    <tr>
     <td>Cellule 2 de tableau</td>
    </tr>
   </tbody>
  </table>
  <!-- The document is valid HTML5 + ARIA + SVG 1.1 + MathML 2.0  -->
 </body>
</html>
```

L'exemple 2-5 présente plusieurs exemples d'inclusions successives d'éléments qui créent une hiérarchie arborescente présentée ci-dessous. L'élément `<body>` a trois enfants directs qui ont chacun à leur tour un ou plusieurs enfants, et ainsi de suite.

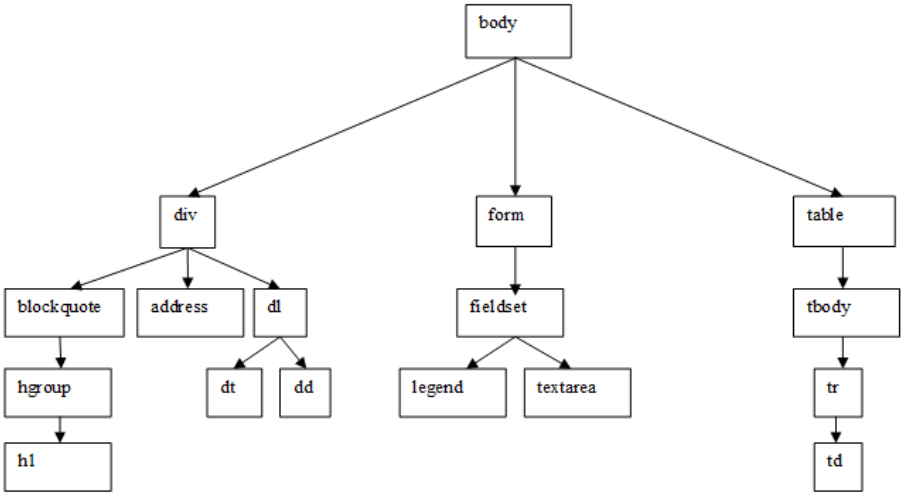


Figure 2-3

Arborescence d'un fichier HTML

Exercices

Exercice 1

La déclaration `DOCTYPE` est-elle obligatoire ? Où doit-elle être placée ? Quelles informations contient-elle ?

Exercice 2

Que faut-il faire pour que le code HTML 5 soit validé ?

Exercice 3

À quel emplacement est défini le jeu de caractères utilisé dans le document ?

Exercice 4

Où est déclaré le nom de l'élément racine du document ? Quel est son rôle et son nom ?

Exercice 5

Quels sont les éléments enfants de l'élément `<html>` ?

12

Dimensionnement, positionnement et transformation des éléments

Les méthodes de mise en page au moyen de cadres ou à l'aide de tableaux (chapitre 6), outre qu'elles sont aujourd'hui considérées comme obsolètes, ne manquent pas de présenter des inconvénients importants. Si les cadres gênaient considérablement l'indexation correcte des pages web, les mises en page à base de tableaux, pour pratiques qu'elles aient l'air à première vue, impliquent des structures trop complexes qui nuisent à la lisibilité de l'organisation et à la maintenance des sites. Nous allons voir dans ce chapitre que tous ces problèmes peuvent être contournés en utilisant conjointement le dimensionnement et le positionnement des éléments qui contiennent les différentes zones d'une page. Ces méthodes apportent enfin des réponses simples pour créer les mises en page les plus diverses, et qui plus est en respectant le principe, fondamental en HTML 5, de séparation du contenu et de la présentation. Nous pouvons par exemple obtenir des présentations différentes à partir du même code HTML 5 en modifiant simplement les styles CSS attachés à ces différents composants.

Le dimensionnement des éléments

Au chapitre 10, dans la définition du modèle de boîtes de CSS, nous avons vu que les propriétés `width` et `height` permettent de déterminer respectivement la largeur et la hauteur d'un élément. Elles s'appliquent à tous les éléments sauf les éléments en ligne

non remplacés et les lignes et groupes de lignes des tableaux. Nous rappelons leurs syntaxes :

```
width: <longueur> | NN% | auto | inherit
height: <longueur> | NN% | auto | inherit
```

Si la définition de la largeur ne pose généralement pas de problème, celle de la hauteur peut engendrer des effets particuliers, voire conflictuels entre les boîtes créées pour les éléments successifs et leurs contenus. L'exemple 12-1 en est une illustration concrète. Le corps du document contient une division `<div>` (repère ❹) qui inclut successivement un titre `<h1>` (repère ❺) et deux paragraphes `<p>` (repères ❻ et ❼). Les paragraphes contiennent pour leur part du texte ordinaire. L'élément `<div>` est suivi d'un second paragraphe ne comprenant que du texte (repère ❽).

Les définitions de styles fixent les dimensions de l'élément `<div>`, qui a une largeur de 80 % et une hauteur de 500 pixels (repère ❶). Les éléments `<p>` ont une largeur de 75 % de celle de leur conteneur (repère ❷), soit 75 % de 80 % (donc 60 % de la largeur de la page) pour le premier, et 75 % de la page pour le second. Ces largeurs sont effectivement respectées par tous les navigateurs (voir la figure 12-1). De même, l'élément `<h1>` (repère ❸) a une largeur réelle de 80 % de celle de son parent `<div>`, donc 80 % de 80 %, soit 64 % de celle de la page.

Mais c'est au niveau de la définition des hauteurs que se posent les problèmes. Nous pouvons remarquer d'emblée que la hauteur de l'élément `<div>` est supérieure à la somme des hauteurs de ses éléments enfants `<p>` (200 pixels chacun) et `<h1>` (50 pixels). Le résultat obtenu visible à la figure 12-1 montre tout d'abord que la hauteur définie pour `<div>` est bien respectée. Celles des deux paragraphes sont bien de 200 pixels, la couleur de fond qui leur est attribuée nous permettant de le vérifier. Il en est de même pour la hauteur de l'élément `<h1>`. En revanche, si le texte du premier paragraphe s'affiche bien dans la boîte assignée à l'élément `<p>`, celui du deuxième déborde, et pire encore il se superpose au contenu du troisième. Le résultat obtenu est évidemment catastrophique au niveau de la présentation.

Exemple 12-1 Les problèmes de dimensionnement en hauteur

```
<!DOCTYPE html>
<html>
 <head>
  <meta http-equiv="Content-type" content="text/html;charset=UTF-8" />
  ➥ <title>Dimensionnement des éléments</title>
  <link rel="shortcut icon" type="images/x-icon" href="../images/favicon.ico" />
  <style type="text/css" >
   body {font-family: Arial sans-serif;font-size: 16px;}
   div{width:80%;height:500px;background-color:#EEE;}❶
   p{width:75%;height:200px;background-color:#BBB;}❷
   h1{width:80%;height:50px;background-color:#888;color:white;}❸
  </style>
 </head>
 <body>
```

```
❹<div>
 ❺<h1>Dimensionnement des éléments</h1>
 ❻<p> In principio creavit Deus caelum et terram terra autem erat inanis et vacua
 ➥ et tenebrae super faciem abyssi et spiritus Dei ferebatur super aquas
 </p>
 ❼<p>Deus duo magna luminaria luminare maius ut praeesset diei et luminare minus
 ➥ ut praeesset nocti et stellas et posuit eas in firmamento caeli . . .</p>
   </div>
 ❽<p> In principio creavit Deus caelum et terram terra autem erat inanis et vacua
 ➥ et tenebrae super faciem abyssi et spiritus Dei ferebatur super aquas . . .
 </p>
</body>
</html>
```

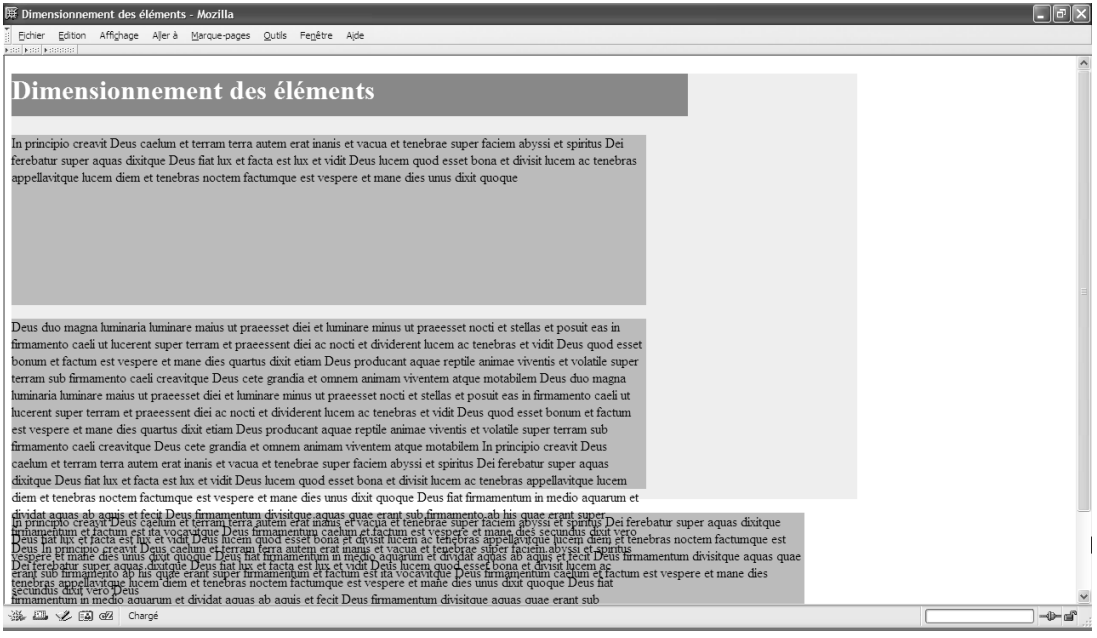


Figure 12-1

Les problèmes dus au dimensionnement

Nous pouvons corriger facilement ce résultat en donnant la valeur `auto` à la propriété `height` des éléments `<div>` et `<p>` (repères ❶ et ❷). L'élément `<style>` devient donc :

```
<style type="text/css" >
 body {font-family: Arial sans-serif;font-size: 16px;}
 div{width:80%;height:auto ❶;background-color:#EEE;}
 p{width:75%;height:auto ❷;background-color:#BBB;}
 h1{width: 80%; height: 50px;background-color:#888;color:white;}
</style>
```

Nous obtenons alors le dimensionnement présenté à la figure 12-2, dans lequel chaque paragraphe occupe la place qui lui est nécessaire pour afficher son contenu.

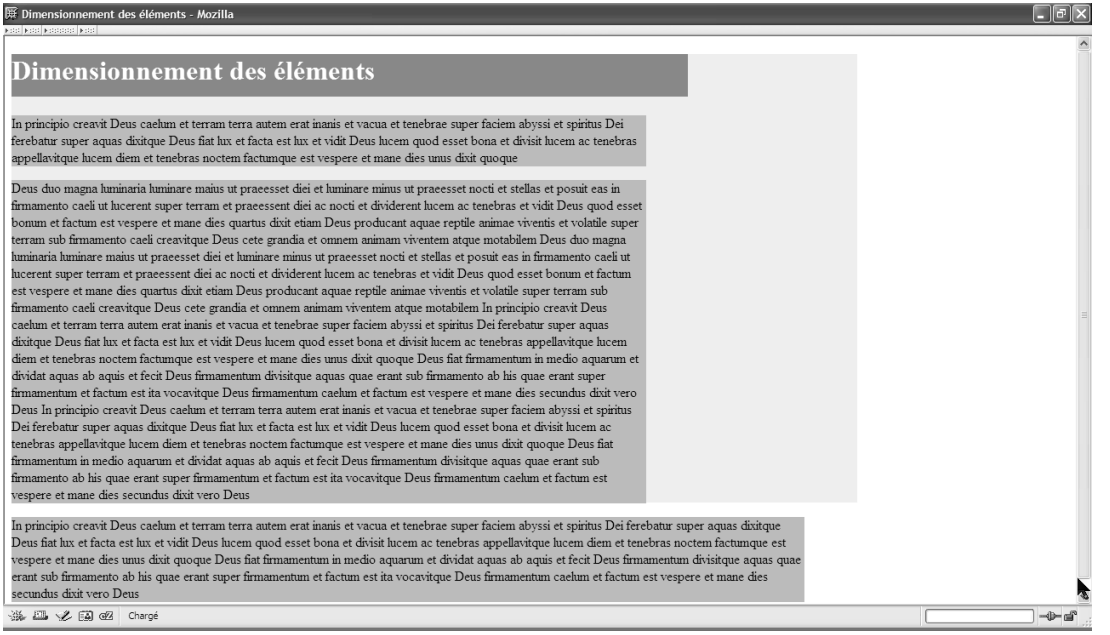


Figure 12-2

Le dimensionnement avec la valeur auto

L'utilisation de la valeur `auto` ne permet pas une mise en page précise des différents éléments dans le document, en particulier si leur contenu est créé dynamiquement, en provenance d'une base de données, suite à une recherche déclenchée par le visiteur par exemple. S'il semble nécessaire au programmeur de définir impérativement une hauteur à chaque élément, il est possible de conserver cette présentation en appliquant la propriété `overflow` pour gérer les éventuels débordements des contenus. Sa syntaxe est la suivante :

```
overflow : visible | hidden | scroll | auto | inherit
```

Voici le rôle imparti à chacune de ses valeurs :

- `visible` : le contenu débordant est affiché ;
- `hidden` : le contenu débordant est caché et donc illisible ;
- `scroll` : des barres de défilement horizontale et verticale apparaissent sur les côtés droit et bas de la boîte de l'élément, ce qui permet d'accéder au contenu débordant. Cette valeur a l'inconvénient de laisser apparaître ces barres même si le contenu ne déborde pas ;
- `auto` : le navigateur fait apparaître les barres de défilement en cas de débordement uniquement.

En modifiant les styles de l'exemple 12-1 de la manière suivante :

```
<style type="text/css" >
 body {font-family: Arial sans-serif;font-size: 16px;}
 div{width:80%;height:500px;background-color:#EEE;}
 p{width:75%;height: 200px;background-color:#BBB;overflow: auto;❶}
 h1{width: 80%; height: 50px;background-color:#888;color:white;}
</style>
```

où nous définissons la propriété `overflow` pour les paragraphes avec la valeur `auto` (repère ❶), nous obtenons le résultat présenté à la figure 12-3 qui montre que la hauteur définie pour les éléments `<p>` est conservée à 200 pixels, mais que le contenu du deuxième paragraphe est lisible en utilisant les barres de défilement. Les intentions de mise en page sont donc préservées et le contenu est entièrement accessible.

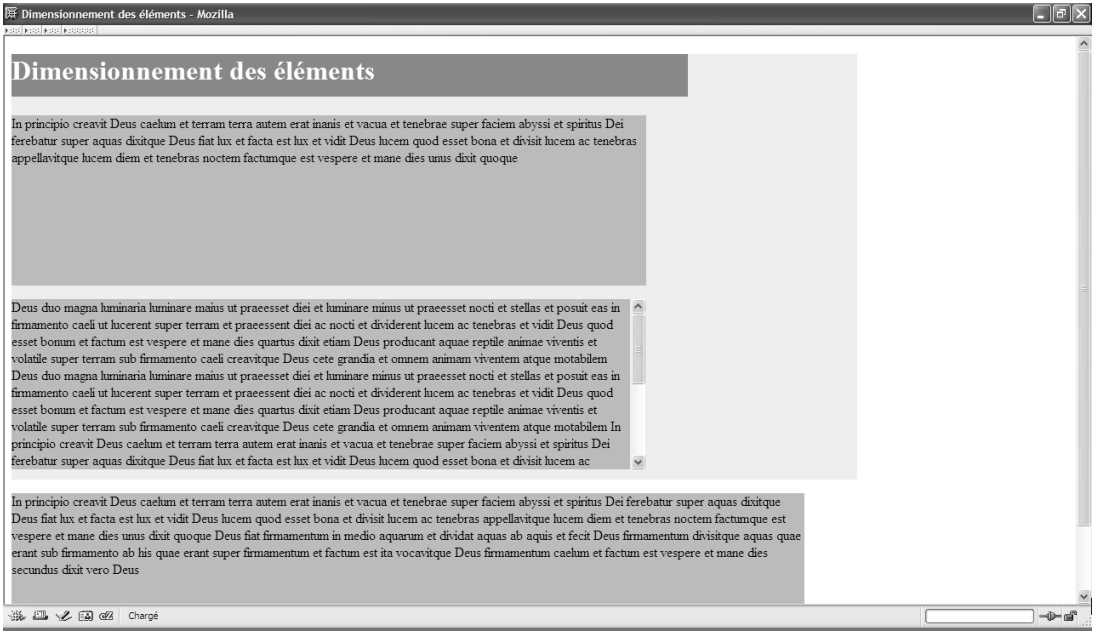


Figure 12-3

Gestion des débordements

Dans les cas précédents, et en gérant les débordements avec la propriété `overflow`, tous les paragraphes ont les mêmes dimensions quels que soient leurs contenus. Dans notre dernier exemple, le premier et le second ont une hauteur de 200 pixels alors que le premier ne contient que quelques lignes. Pour pouvoir intervenir sur les hauteurs des éléments sans pour autant les fixer de manière absolue, nous pouvons définir une hauteur minimale et une hauteur maximale pour tous les éléments, à l'exception des éléments en ligne non remplacés et des éléments de tableau.

Nous disposons pour cela des propriétés `min-height` et `max-height` dont la syntaxe est la suivante :

```
min-height: <longueur> | NN% | inherit
max-height: <longueur> | NN% | none |inherit
```

Le paramètre `<longueur>` est donné comme d'habitude par un nombre et une unité, et les pourcentages sont précisés en référence à la hauteur du bloc conteneur. La valeur `none` (qui est la valeur par défaut pour la hauteur maximale) indique que la valeur limite est celle du bloc conteneur.

De même, une largeur minimale et une largeur maximale peuvent être indiquées pour les mêmes éléments à l'aide des propriétés `min-width` et `max-width` dont les syntaxes sont similaires aux précédentes :

```
min-width: <longueur> | NN% | inherit
max-width: <longueur> | NN% | none |inherit
```

L'exemple 12-2 nous permet d'affiner les mises en page obtenues dans les exemples précédents. L'élément `<div>` se voit affecté une largeur maximale de 900 pixels et une hauteur maximale de 800 pixels (repère ❶). Les paragraphes ont une hauteur minimale de 80 pixels et maximale de 250 pixels (repères ❷ et ❸), ainsi qu'une largeur comprise entre 500 et 600 pixels (repères ❹ et ❺). Les éventuels débordements sont gérés en utilisant la propriété `overflow` (repère ❻). L'élément `<h1>` a pour sa part une largeur comprise entre 400 et 500 pixels au maximum (repères ❼ et ❽). Comme nous pouvons le constater sur la figure 12-4, le contenu du titre `<h1>` est affiché sur deux lignes car sa largeur est limitée à 400 pixels. Le premier élément `<p>` (repère ❾) occupe juste la hauteur nécessaire à son contenu limité à quatre lignes. En revanche, le deuxième (repère ❿) a un contenu beaucoup plus long et il apparaît avec une hauteur de 250 pixels, ce qui est sa limite supérieure ; comme son contenu déborde, des barres de défilement apparaissent automatiquement. Quant au troisième paragraphe (repère ⓫), il a le même comportement que le premier et sa hauteur est exactement adaptée à son contenu en respectant les contraintes de hauteur.

Exemple 12-2 Largeurs et hauteurs minimales et maximales

```
<!DOCTYPE html>
<html>
 <head>
  <meta http-equiv="Content-type" content="text/html;charset=UTF-8" />
  <title>Dimensions minimales et maximales des éléments</title>
  <link rel="shortcut icon" type="images/x-icon" href="../images/favicon.ico" />
  <style type="text/css" >
   body {font-family: Arial sans-serif;font-size: 16px;}
   div{max-width:900px;max-height:800px;background-color:#EEE;} ❶
   p{min-height:80px ❷ ;max-height: 250px ❸ ;min-width:500px ❹;
   ➥ max-width:600px ❺; background-color:#BBB;overflow: auto ❻;}
   h1{min-width: 400px ❼; max-width: 500px ❽;background-color:#888;color:white;}
  </style>
 </head>
 <body>
```

```
<div>
 <h1>Dimensions mini et maxi des éléments HTML 5</h1>
❾ <p> In principio creavit Deus caelum et terram terra autem erat inanis et vacua
   ➥ et tenebrae super faciem abyssi et spiritus Dei ferebatur super aquas
   dixitque Deus fiat lux et facta est lux et vidit Deus lucem quod esset bona et
   ➥ divisit lucem ac tenebras appellavitque lucem diem et tenebras noctem
   ➥ factumque est vespere et mane dies quartus dixit etiam Deus producant</p>

❿ <p>Deus duo magna luminaria luminare maius ut praeesset diei et luminare minus
   ➥ ut praeesset nocti et stellas et posuit eas in firmamento caeli ut lucerent
   ➥ super terram et praeessent diei ac nocti et dividerent lucem ac tenebras et
   ➥ vidit Deus quod esset bonum et factum est vespere et mane dies …</p>
</div>
⓫ <p> In principio creavit Deus caelum et terram terra autem erat inanis et vacua
   ➥ et tenebrae super faciem abyssi et spiritus Dei ferebatur super aquas
   ➥ est ita vocavitque Deus firmamentum caelum et factum est vespere et mane dies
   ➥ secundus dixit vero Deus  </p>
 </body>
</html>
```

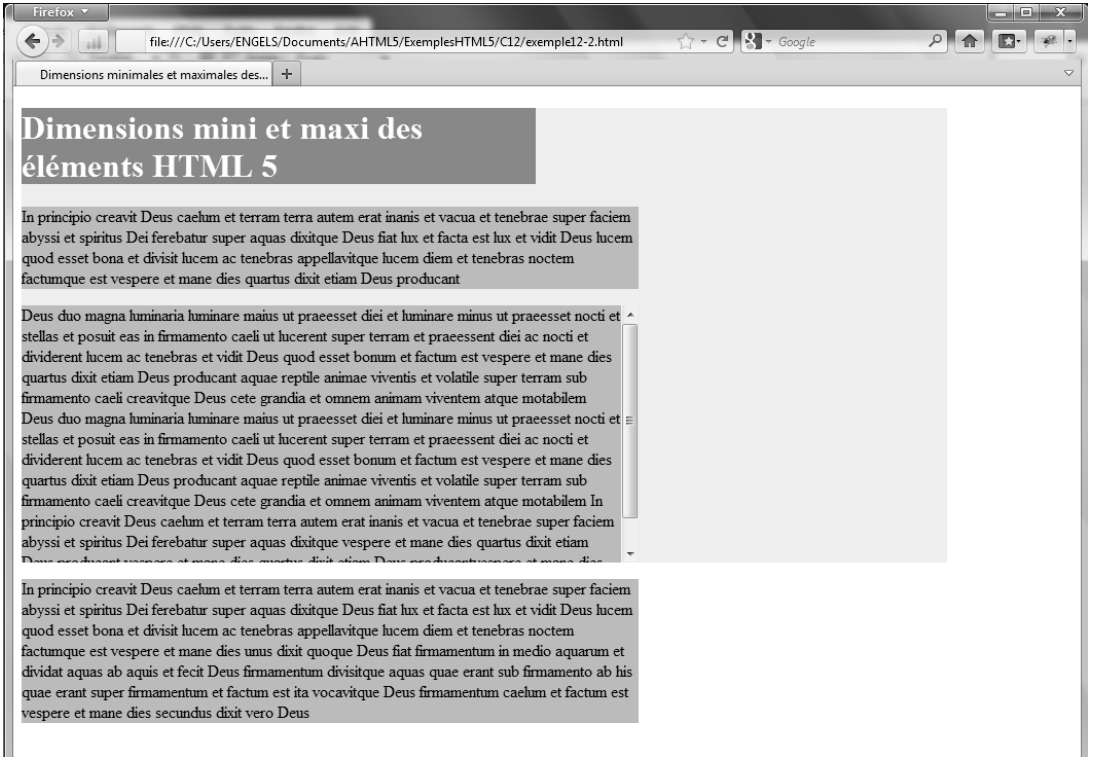


Figure 12-4

Les dimensions minimales et maximales

Le rendu des éléments

Dans la première partie de cet ouvrage, nous avons passé en revue l'ensemble des éléments HTML 5 et nous avons vu que chacun d'eux est associé par défaut à une présentation spécifique (à l'aide d'une feuille de styles par défaut incluse dans les navigateurs : voir l'annexe B). Tous les éléments peuvent donc être classés en grands groupes de mise en page comme les blocs, les éléments en ligne, les listes ou les tableaux avec les caractéristiques qui s'y rattachent. Nous pouvons intervenir sur l'appartenance de chaque élément à un de ces groupes et modifier le rendu d'un élément en fonction des besoins. Un élément `<h1>` peut, par exemple, être transformé en élément en ligne ou en élément de liste alors qu'il est par défaut de niveau bloc.

Ces modifications sont opérées au moyen de la propriété `display` dont la syntaxe est la suivante :

```
display : none |inline | block | list-item |table | inline-table | table-row-group |
➥ table-header-group | table-footer-group | table-row | table-column-group | table-
➥ column | table-cell | table-caption | inherit
```

Sa valeur par défaut est `inline` mais la feuille de styles par défaut la modifie et donne à chaque élément le style que nous lui connaissons. Ce sont ces valeurs par défaut sur lesquelles nous intervenons. Les principales valeurs qu'elle peut prendre ont la signification suivante :

- `none` : l'élément n'est pas affiché et la boîte qui lui est associée n'est pas créée. Tout se passe comme s'il n'existait pas dans le code HTML 5 ;
- `inline` : l'élément devient du type en ligne (comme `<span>`) ;
- `block` : l'élément devient du type bloc (comme `<h1>`, `<p>`, `<div>`…) ;
- `list-item` : l'élément devient du type liste (équivalent de `<li>`).

Les autres valeurs sont rarement utilisées en HTML 5, mais plutôt réservées à la création de styles destinés à des documents XML pour induire des comportements identiques à ceux des éléments de tableau HTML 5. Le tableau suivant donne les correspondances entre les valeurs de la propriété `display` et l'élément HTML 5 dont le comportement est induit par celles-ci.

Tableau 12-1. Correspondances entre les valeurs de la propriété display et les éléments HTML 5

Valeur	Élément	Valeur	Élément
`table`	`<table>`	`table-row-group`	`<tbody>`
`table-header-group`	`<thead>`	`table-footer-group`	`<tfoot>`
`table-row`	`<tr>`	`table-column-group`	`<colgroup>`
`table-column`	`<col />`	`table-cell`	`<td>` ou `<th>`
`table-caption`	`<caption>`	`inline-table`	`<table>` dont la propriété `display` vaut `inline`

L'exemple 12-3 offre une illustration de la propriété `display`. La page comprend un paragraphe `<p>` qui inclut quatre éléments `<span>` (repère ❹). Ces éléments, qui sont normalement de type en ligne, sont affichés sous forme de liste en donnant la valeur `list-item` à la propriété `display` (repère ❶). La page contient également une liste non ordonnée (repère ❺). En donnant la valeur `inline` à `display` pour les éléments `<li>` (repère ❷), ils constituent un menu sur une seule ligne. La page contient enfin une division (repère ❻) incluant une image (repère ❾), deux titres `<h1>` (repères ❿ et ⓫) et un paragraphe (repère ⓬). Pour ces derniers, la propriété `display` prend la valeur `inline` (repère ❸). Le contenu des titres est donc affiché comme du texte en ligne dans le paragraphe. Quant à l'image incluse dans `<div>`, le gestionnaire d'événements `onclick` permet de la faire disparaître quand le visiteur clique sur la division en donnant, à l'aide de code JavaScript, la valeur `none` à la propriété `display` (repère ❼) ; en gérant l'événement `ondblclick`, un double-clic la fait réapparaître en lui octroyant la valeur `inline` (repère ❽).

Exemple 12-3 Modification du rendu des éléments

```
<!DOCTYPE html>
<html>
 <head>
  <meta http-equiv="Content-type" content="text/html;charset=UTF-8" />
  <title>Rendu des éléments</title>
  <link rel="shortcut icon" type="images/x-icon" href="../images/favicon.ico" />
  <style type="text/css" >
   body,h1,p{font-size: 24px;}
   span{display: list-item; } ❶
   li{display: inline;border: solid 1px black;} ❷
      h1,p{display: inline;} ❸
  </style>
 </head>
 <body>
 ❹ <p><span> HTML 5 </span> <span> CSS 3 </span> <span> JavaScript </span> <span>
    ➥ PHP 5 </span>  </p>
 ❺ <ul>
    <li><a  href="lien1.html" ></a>..HTML 5 .. </li>
    <li><a  href="lien2.html" ></a>..CSS 3.. </li>
    <li><a  href="lien3.html" ></a>..JavaScript.. </li>
    <li><a  href="lien4.html" ></a>..PHP 5.. </li>
  </ul>
 ❻ <div id="division" ❼ onclick="document.getElementById('couv').style.
➥ display='none'"
 ❽ ondblclick="document.getElementById('couv').style.display='inline'" >
   ❾ <img id="couv"  src="couvhtml5_gd.jpg" alt="HTML 5 et  CSS 3" height="130"
   ➥ width="107" /> Les langages du Web :
   ❿ <h1> HTML 5    </h1>
   ⓫ <h1> CSS 3 : </h1>
  ⓬ <p> In principio creavit Deus caelum et terram terra autem erat inanis et vacua
  ➥ et tenebrae super faciem abyssi et spiritus Dei ferebatur super aquas dixitque
  ➥ Deus fiat lux et facta est lux et vidit Deus lucem quod esset bona et divisit
```

```
    ➥ lucem ac tenebras appellavitque lucem diem et tenebras noctem factumque est
    ➥ vespere et mane dies unus dixit quoque Deus fiat firmamentum in medio aquarum et
    ➥ dividat aquas ab aquis et fecit </p>
   </div>
  </body>
</html>
```

La figure 12-5 montre le résultat obtenu avant la disparition de l'image.

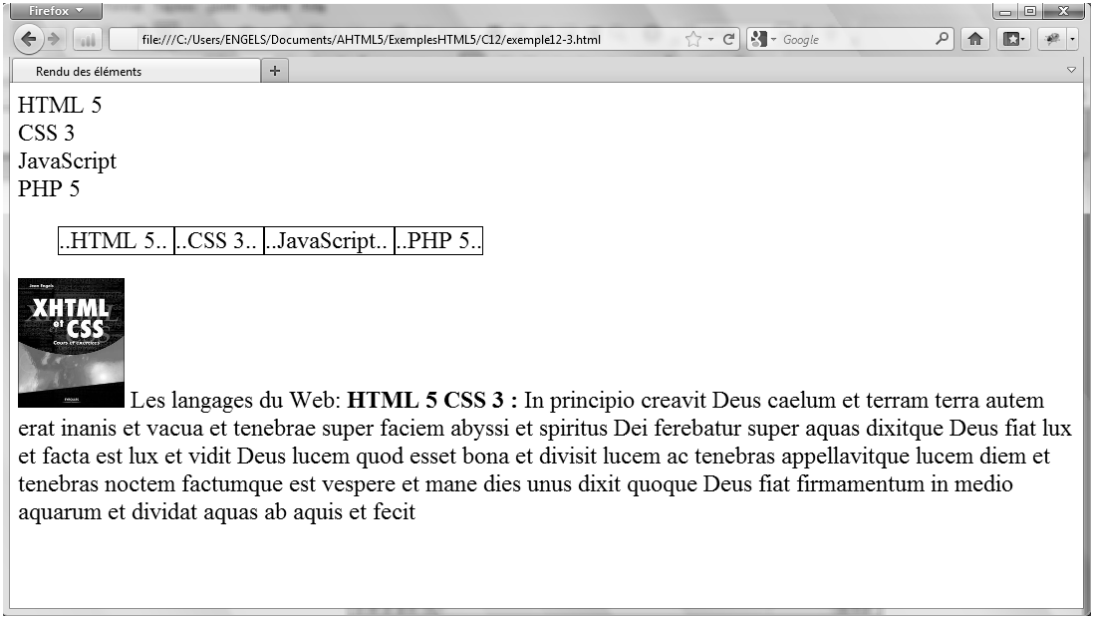


Figure 12-5

Les modifications du rendu normal des éléments

Le positionnement des éléments

Le principe du positionnement permet de définir l'emplacement des boîtes générées pour chaque élément HTML 5 présent dans le code de la page. Il existe plusieurs schémas de positionnement en CSS 2.1.

- Le positionnement selon le flux normal inclut le positionnement par défaut opéré par les navigateurs sans définition de styles particuliers. Avec ce positionnement, les éléments `<p>`, `<div>` ou `<h1>`, par exemple, sont associés à une boîte de bloc, et les éléments `<img />` ou `<span>` à une boîte en ligne. Il inclut également le positionnement relatif des éléments de type bloc ou en ligne.
- Le positionnement flottant : dans ce cas, la boîte de l'élément est générée comme dans le flux normal. Elle conserve pour exemple les dimensions qui lui ont été attribuées,

soit par ses attributs, soit par les propriétés `width` et `height`. En revanche, cette boîte est déplacée afin d'être positionnée le plus haut et le plus à gauche possible dans le contenu.

- Le positionnement absolu : avec ce type de positionnement, le bloc généré par l'élément devient complètement indépendant du flux normal. Sa position est en particulier sans aucun rapport avec son ordre d'apparition dans le code HTML 5. Des propriétés particulières doivent être définies pour placer la boîte de l'élément par rapport à son contenu, qu'il s'agisse de `<body>` ou d'un autre élément.
- Le positionnement fixe : dans ce cas, un élément occupe une position fixe dans la fenêtre du navigateur et ne défile pas avec le reste de la page.

Le flottement

Le concept du modèle de boîtes vu au chapitre 11 suppose que les blocs apparaissent dans la page les uns à la suite des autres, de haut en bas, selon leur ordre d'écriture dans le code HTML 5. De même, un élément remplacé, comme une image incluse dans un paragraphe au milieu d'un texte, est affiché comme un sous-bloc en provoquant un retour à la ligne avant et après l'image. Si sa largeur est inférieure à celle du paragraphe, cela entraîne l'apparition d'une zone blanche à sa droite, car elle est alignée à gauche par défaut. Cela crée un effet évidemment disgracieux dans la page. Il en est de même pour un élément de bloc, par exemple un tableau dont la largeur est faible par rapport à celle de son conteneur.

Pour améliorer cet état de fait, nous pouvons rendre n'importe quel élément flottant. Quand un élément est déclaré flottant, le contenu des autres éléments voisins se dispose autour de lui comme l'eau s'écoule autour d'un rocher placé au milieu d'une rivière. L'espace laissé libre autour de cet élément est comblé, créant ainsi une meilleure occupation de la surface de la page.

Quand un élément est déclaré flottant, la boîte qui lui correspond est déplacée vers la gauche ou la droite de son conteneur, selon la valeur choisie pour le flottement. L'alignement vertical est tel que la boîte est déplacée sur le haut de la ligne qui contient l'élément, ou sur le bas du bloc qui le précède. On procède à la définition du flottement d'un élément au moyen de la propriété `float` qui peut s'appliquer à tous les éléments HTML 5 et dont la syntaxe est la suivante :

```
float:left|right|none|inherit
```

Les valeurs qu'elle peut prendre ont la signification suivante :

- `left` : la boîte de l'élément est déplacée en haut et à gauche de son conteneur ;
- `right` : la boîte de l'élément est déplacée en haut et à droite ;
- `none` : la boîte ne flotte pas (c'est la valeur par défaut) ;
- `inherit` : la boîte hérite du comportement de son élément parent (ce qui n'est pas le cas par défaut).

Pour qu'un élément soit flottant, il est nécessaire qu'il soit dimensionné explicitement avec la propriété `width`, ou implicitement par ses dimensions intrinsèques (celles d'une image par exemple), ou par ses attributs `width` et `height` dans le cas d'un élément `<img />` ou `<object>`. Dans le cas contraire, le navigateur risque de l'afficher avec une largeur minimale, si ce n'est nulle.

Si un élément flottant possède des marges, ces dernières ne fusionnent pas avec celles des éléments voisins, mais elles s'ajoutent à celles-ci. Quand plusieurs éléments sont flottants du même côté, celui qui apparaît en seconde position dans le code peut être amené à buter sur le bord droit ou gauche du premier selon que la propriété `float` a respectivement la valeur `left` ou `right`.

Dans l'exemple 12-4, la page contient un élément `<div>` (repère ❹) qui inclut du texte brut, une image (repères ❷ et ❺), à nouveau du texte puis un paragraphe (repères ❸ et ❻). La division a une couleur de fond, des marges de 15 pixels et un alignement justifié (repère ❶). L'image est positionnée flottante à gauche et a des marges de 20 pixels. Le paragraphe est flottant à droite, dimensionné avec une largeur de 200 pixels, et est muni d'une bordure qui permet de bien le distinguer du reste du texte. La figure 12-6 montre le résultat obtenu. Nous y constatons que les éléments flottants n'apparaissent que relativement à l'ordre dans lequel ils figurent dans le code HTML 5. Par exemple, l'image n'apparaît qu'une fois que la ligne dans laquelle l'élément `<img />` est situé a été complétée par du texte qui se situe après cet élément.

Exemple 12-4 Les éléments flottants

```
<!DOCTYPE html>
<html>
 <head>
  <meta http-equiv="Content-type" content="text/html;charset=UTF-8" />
  <title>Les éléments flottants</title>
  <link rel="shortcut icon" type="images/x-icon" href="../images/favicon.ico" />
  <style type="text/css" >
   div{background-color: #EEE;margin: 15px;text-align: justify;} ❶
   img{float:left;margin:20px;} ❷
   p.droit{float: right;border:2px solid red;width: 200px;margin:10px;
   ➥ padding: 10px;} ❸
  </style>
 </head>
 <body>
 ❹ <div>HTML 5 ET CSS : In principio creavit Deus caelum et terram terra autem erat
  ➥ inanis et vacua et tenebrae super
  faciem abyssi et spiritus Dei ferebatur super aquas dixitque Deus fiat lux et facta
  ➥ est lux et vidit Deus
  ❺ <img src="couvhtml5_gd.jpg" alt="HTML 5 et CSS 3" />
  lucem quod esset bona et divisit lucem ac tenebras appellavitque lucem
  diem et tenebras noctem factumque est vespere et mane dies unus dixit quoque Deus
  ➥ fiat firmamentum in medio aquarum et……
```

```
❻ <p class="droit">Le W3C <br />aquas quae erant sub firmamento ab his quae erant
➥ super firmamentum et factum est ita vocavitque Deus firmamentum caelum et factum
➥ est vespere et mane dies secundus dixit vero Deus congregentur aquae quae sub
➥ caelo sunt in locum unum et appareat  </p>
in firmamento caeli et inluminent terram et factum est ita fecitque Deus duo magna
➥ luminaria luminare maius ut praeesset diei et luminare minus ut praeesset nocti
➥ et stellas et posuit eas in firmamento caeli ut lucerent super terram et
➥ praeessent diei ac nocti et dividerent lucem ac tenebras et  </div>
</body>
</html>
```

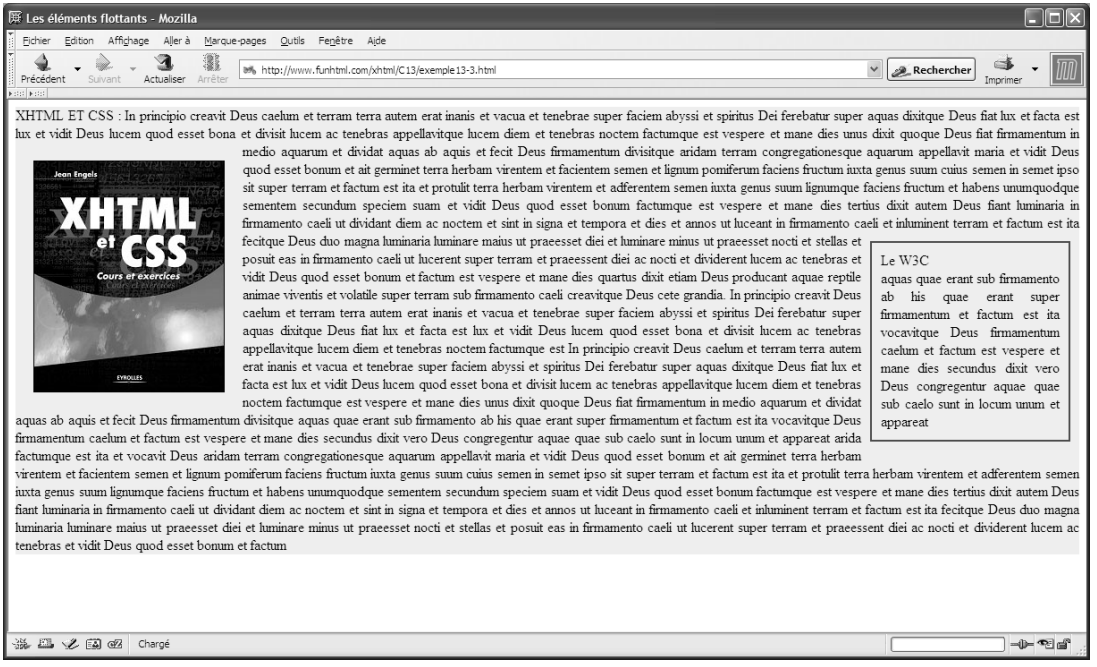


Figure 12-6

Le flottement des éléments

L'exemple suivant permet d'appréhender les subtilités du positionnement flottant dans des cas particuliers. L'élément `<div>` (repère ❸) inclus dans la page contient maintenant trois images incorporées au milieu d'un texte (repères ❹, ❺ et ❻). Vient ensuite un paragraphe ne contenant que du texte (repère ❼). Les deux premières images sont flottantes à gauche (repère ❶), et la troisième à droite en utilisant la classe `img.droit` (repère ❷). Seule la deuxième image est dimensionnée explicitement avec ses attributs `height` et `width`, les autres ayant les dimensions originales de l'image.

Exemple 12-5 Cas particuliers de flottement

```
<!DOCTYPE html>
<html>
 <head>
  <meta http-equiv="Content-type" content="text/html;charset=UTF-8" />
  ➥ <title>Les éléments flottants</title>
  <link rel="shortcut icon" type="images/x-icon" href="../images/favicon.ico" />
  <style type="text/css" >
   div{background-color: #EEE;}
   img{float:left;margin:20px;}❶
   img.droit{float: right;}❷
   p{background-color: #DD2;}
  </style>
 </head>
 <body>
   ❸<div>HTML 5 ET CSS : In principio creavit Deus caelum et terram terra autem
   ➥ erat inanis et vacua et tenebrae super faciem abyssi et spiritus Dei ferebatur
   ➥ super aquas dixitque Deus fiat lux et facta est lux et vidit Deus
   ❹<img src="couvhtml5_gd.jpg" alt="HTML 5 et CSS 3" />
   lucem quod esset bona et divisit lucem ac tenebras appellavitque lucem diem et
   ➥ tenebras noctem factumque est vespere et mane dies unus dixit quoque Deus
   ➥ fiat . . .
   ❺<img src="couvhtml5_gd.jpg" alt="HTML 5 et CSS 3" height="130" width="107" />
   herbam virentem et adferentem semen iuxta genus suum lignumque faciens fructum et
   ➥ habens unumquodque sementem secundum speciem suam et vidit Deus . . .
   ❻<img class="droit" src="couvhtml5_gd.jpg" alt="HTML 5 et CSS 3" />ut praeesset
   ➥ nocti et stellas et posuit eas in firmamento caeli ut lucerent super terram et
   ➥ praeessent diei ac nocti et dividerent lucem ac tenebras et vidit Deus . . .
  </div>
   ❼<p>In principio creavit Deus caelum et terram terra autem erat inanis et vacua
   ➥ et tenebrae super faciem abyssi et spiritus Dei ferebatur super aquas dixitque
   ➥ Deus fiat lux et facta est lux et vidit Deus . . . </p>
 </body>
</html>
```

On peut le vérifier à la figure 12-7 ; la première image est positionnée comme dans l'exemple précédent, et la deuxième flottante à gauche vient se ranger le plus à gauche possible, selon les principes énoncés plus haut. Mais comme l'espace est occupé par la première, elle vient se placer contre le bord droit de cette dernière, et non plus sur le bord gauche de son conteneur. Nous pouvons remarquer à cette occasion que les marges de chacune des images sont conservées et ne fusionnent pas. Le cas de la troisième image flottante à droite est le plus particulier. En effet, l'élément `<div>` ne pouvant la contenir en entier, elle déborde et empiète sur le paragraphe qui suit et se comporte comme si elle était flottante dans ce paragraphe, le texte de ce dernier l'entourant.

Figure 12-7

Cas particuliers de flottement

Empêcher le flottement

Nous pouvons empêcher le type de comportement de la troisième image qui flotte sur le paragraphe `<p>` en appliquant pour ce dernier la propriété `clear`. Ce procédé n'est possible que pour les éléments de bloc, mais il permet d'éviter le flottement d'un élément sur la gauche ou la droite de celui pour lequel la propriété est définie. Ainsi, l'image a la place nécessaire à son affichage et le texte du paragraphe est repoussé vers le bas, laissant apparaître un espace vide. Cela peut constituer un choix de présentation dans certains cas, mais la solution précédente est en l'occurrence plus compacte. Voici la syntaxe de la propriété `clear` :

```
clear: none | left | right | both | inherit
```

Les valeurs qu'elle peut prendre ont pour signification :

- `none` : le flottement est autorisé ;
- `left` : le flottement d'un élément sur la gauche est interdit ;
- `right` : le flottement d'un élément sur la droite est interdit ;
- `both` : le flottement d'un élément est interdit à gauche et à droite.

En conservant le code HTML 5 de l'exemple 12-5 et en modifiant seulement les styles de la manière suivante (repère ❶) :

```
<style type="text/css" >
 div{background-color: #EEE;}
 img{float:left;margin:20px;}
 img.droit{float: right;}
 p{background-color: #DD2;clear:right;}❶
</style>
```

où la propriété `clear` est définie avec la valeur `right` pour l'élément `<p>`, nous obtenons le résultat présenté à la figure 12-8 qui montre que la troisième image ne flotte plus sur le paragraphe.

Figure 12-8

Suppression du flottement sur le côté d'un élément

Positionnement relatif

Dans le positionnement relatif, les navigateurs analysent un document et déterminent un emplacement pour chacun des éléments comme dans le flux normal, puis déplacent ceux qui ont un positionnement relatif par rapport à la position qu'ils auraient dû occuper, mais en ne déplaçant pas les autres éléments. En conséquence, l'emplacement initial reste vide et il en résulte des chevauchements.

En positionnement relatif, l'ordre d'écriture des éléments est important car il conditionne l'emplacement de chaque élément (avant le déplacement relatif) et l'ordre de superposition en cas de chevauchement des boîtes ; la dernière écrite dans le code se superpose à celle de l'élément placé avant.

Le positionnement relatif est spécifié au moyen de la propriété `position`, munie de la valeur `relative`. Prise isolément, cette propriété n'a aucun effet. Il faut ensuite préciser le décalage voulu au moyen des propriétés `left`, `top`, `right` et `bottom`, dont les significations sont les suivantes :

- `left` : décale l'élément vers la droite (si sa valeur est positive) ou vers la gauche (si sa valeur est négative) ;
- `top` : décale l'élément vers le bas (si sa valeur est positive) ou vers le haut (si sa valeur est négative) ;
- `right` : décale l'élément vers la gauche (si sa valeur est positive) ou vers la droite (si sa valeur est négative) ;
- `bottom` : décale l'élément vers le haut (si sa valeur est positive) ou vers le bas (si sa valeur est négative).

Les quatre propriétés `left`, `top`, `right` et `bottom` ont la même syntaxe. Par exemple :

```
left: <longueur> | NN% | auto | inherit
```

Le paramètre `<longueur>` est donné comme à l'habitude par un nombre et une unité ; les pourcentages se réfèrent aux dimensions du bloc conteneur. Avec le mot-clé `auto`, la valeur de la propriété est calculée en fonction de la valeur de celle qui lui est complémentaire (les couples `left`/`right` et `top`/`bottom` sont complémentaires) de la manière suivante :

- si les deux propriétés complémentaires ont la valeur `auto`, leur valeur calculée est `0` ;
- si l'une vaut `auto` et que l'autre a une valeur numérique explicite, la première prend la valeur opposée à celle de la seconde ;
- si les deux propriétés complémentaires sont définies avec des valeurs numériques (donc différentes de `auto`) et que ces deux valeurs ne sont pas opposées, la valeur qui l'emporte dépend du sens de lecture du texte définie par l'attribut `dir`. S'il vaut `ltr`, c'est la propriété `left` qui l'emporte, et `right` prend la valeur opposée ; sinon, quand il vaut `rtl`, c'est `right` qui l'emporte.

L'exemple 12-6 illustre le positionnement relatif. L'élément `<div>` (repère ❺) inclut un paragraphe qui contient du texte brut, puis trois images (repères ❼, ❽ et ❾) dans un ordre donné. Si aucun de ces éléments n'est positionné, nous obtenons le résultat habituel présenté à la figure 12-9. Si nous plaçons chacune de ces images en leur appliquant respectivement les classes `img.un`, `img.deux` et `img.trois`, nous obtenons les déplacements relatifs suivants :

- image 1 : déplacement de 30 pixels vers le bas (`top` vaut `30 px`) et de 40 pixels vers la droite (`left` vaut `40 px`) (repère ❶) ;
- image 2 : déplacement de 50 pixels vers le haut (`top` a une valeur négative de `- 50 px`) et de 40 pixels vers la gauche (`left` a une valeur négative de `- 40 px`) (repère ❷) ;

- image 3 : déplacement de 80 pixels vers le haut (`top` a une valeur négative de `- 80 px`) et de 40 pixels vers la droite (`left` vaut `40 px`) (repère ❸).

Le paragraphe est quant à lui déplacé de 20 pixels vers le bas (`bottom` a une valeur négative de `- 20 px` équivalente au style `top:20 px`) et de 6 % de la largeur de son conteneur (l'élément `<div>`) vers la droite (repères ❹ et ❻). Notons ici que les valeurs négatives de pourcentage ne fonctionnent pas.

Figure 12-9

La page sans positionnement

Exemple 12-6 Le positionnement relatif

```
<!DOCTYPE html>
<html>
 <head>
  <meta http-equiv="Content-type" content="text/html;charset=UTF-8" />
  <title>Positionnement relatif</title>
  <link rel="shortcut icon" type="images/x-icon" href="../images/favicon.ico" />
  <style type="text/css" >
   div{background-color: #EEE;}
   img{margin:20px;}
   img.un{position:relative;top:30px;left:40px;}❶
   img.deux{position:relative; top:-50px; left:-40px;}❷
   img.trois{position:relative; top:-80px; left:40px;}❸
```

```
    p{background-color: #DD2;position:relative; bottom:-20px; left: 6% ;}❹
   </style>
  </head>
  <body>
   ❺<div>
    ❻<p><big>HTML 5 </big>: In principio creavit Deus caelum et terram terra autem
    ➥ erat inanis et vacua et tenebrae super faciem abyssi et spiritus Dei
    ➥ ferebatur super aquas dixitque Deus fiat lux et facta est lux et . . . </p>
    ❼<img src="couvhtml5_gd.jpg" alt="HTML 5 et CSS 3" class="un"/>
    ❽<img src="couvhtml5_gd.jpg" alt="HTML 5 et CSS 3" height="130" width="107"
    ➥ class="deux"/>
    ❾<img class="trois" src="couvhtml5_gd.jpg" alt="HTML 5 et CSS 3" />
   </div>
  </body>
 </html>
```

La figure 12-10 donne le résultat obtenu après le positionnement relatif. Nous constatons que les éléments déplacés relativement à leur position dans le flux normal peuvent se superposer aux autres éléments voisins, la deuxième image étant placée au-dessus de la première, et la troisième se superposant au texte du paragraphe.

Figure 12-10

La page avec positionnement

Positionnement absolu

Dans le positionnement absolu, qui peut s'appliquer à tous les éléments, la boîte créée pour l'élément concerné n'apparaît plus dans le flux normal du document. Autrement dit, si un bloc `<div>` est positionné de manière absolue, il peut être écrit n'importe où dans le document HTML 5 sans que cela ne modifie la position qui va lui être assignée avec un style CSS. Le positionnement d'un élément est effectué par rapport au bloc de son conteneur (il s'agit souvent de l'élément `<body>`, mais pas nécessairement). Chaque bloc d'un élément positionné de manière absolue devient à son tour le conteneur de ses éléments enfants. Si ces éléments sont eux-mêmes en position absolue, ils le sont par rapport à leur bloc parent direct et non par rapport au bloc qui contient leur parent. Nous pouvons donc créer des blocs `<div>` et les positionner, puis écrire normalement leur contenu.

L'indépendance de la position par rapport au flux normal fait que deux éléments en position absolue peuvent occuper la même zone dans leur bloc parent commun. Dans ce cas, aucun d'eux ne repousse l'autre comme il en va pour les éléments flottants. Le bloc d'un élément peut alors recouvrir l'autre en fonction de leur ordre d'empilement défini par la propriété `z-index`. Cette propriété peut alors être utilisée pour créer des effets dynamiques d'apparition et de disparition gérés par des scripts JavaScript.

Comme le positionnement relatif, le positionnement absolu est défini en utilisant encore la propriété `position`, mais en lui donnant cette fois la valeur `absolute` (ou `fixed` sur laquelle nous reviendrons par la suite). Il faut ensuite définir la position de l'élément par rapport à son conteneur à l'aide des propriétés `left`, `top`, `right` et `bottom`, qui définissent la position des bords de l'élément respectivement par rapport aux bords gauche, haut, droit et bas du conteneur. Nous ne définissons généralement que deux de ces propriétés, le plus souvent `left` et `top` (les propriétés symétriques `right` et `bottom` prenant une valeur opposée), et la boîte de l'élément doit être dimensionnée avec les propriétés `width` et `height`. L'utilisation conjointe des propriétés `position` et `float` est impossible, et si c'est le cas la propriété `float` prend automatiquement la valeur `none`.

Dans les exemples suivants, nous allons étudier divers cas de mise en page correspondant à des types de présentation couramment rencontrés sur le Web.

Le premier cas présenté dans l'exemple 12-7 consiste à diviser la page en deux zones horizontales. La zone supérieure est un bandeau contenant le titre du site et un menu composé de liens vers les différentes pages. Pour permettre une navigation aisée, toutes les pages contiennent ce même bandeau dont il suffit de copier le code et les styles qui lui sont associés dans chacune des pages. Il est contenu dans un élément `<div>` muni d'un attribut `id` (repère ❺) ; il inclut également un titre `<h1>` (repère ❻) et le menu créé par une liste non ordonnée `<ul>` (repère ❼) dont chaque item contient un lien vers les différentes pages.

Ce premier élément `<div>` est positionné de manière absolue en haut et à gauche de la page en définissant les propriétés `top` et `left` avec la valeur `0`. Il est dimensionné à 100 % en largeur et à 110 pixels en hauteur (repère ❶). La propriété `display` appliquée à l'élément `<li>` permet d'obtenir le menu des liens en ligne sous forme horizontale (repère ❸). Les items sont, de plus, munis de bordures pour en améliorer l'aspect.

La zone de contenu est également incluse dans un second élément `<div>` qui contient un titre `<h1>`, un paragraphe `<p>` et une image (repères ❹ et ❽), ces éléments n'étant pas positionnés. Comme la première, cette zone est dimensionnée à 100 % de la largeur de la page et positionnée de manière absolue, en étant placée à 110 pixels du bord supérieur de la page (correspondant à la hauteur du bandeau) et collée à son bord gauche en mettant la propriété `left` à la valeur `0` (repère ❷).

Exemple 12-7 Positionnement en deux blocs horizontaux

```
<!DOCTYPE html>
<html>
 <head>
  <meta http-equiv="Content-type" content="text/html;charset=UTF-8" />
  <title> La référence des éléments  HTML 5 </title>
  <style type="text/css">
   body{font-size: 18px;}
   h1{font-size: 2em;margin-top: 5px;}
   div#menu {position: absolute ; width:100%; height: 110px; left:0px; top:0;
   ➥ background-color:rgb(255,102,5);color:white; }❶
   div#corps { position: absolute ;width:100%; left:0; top:110px;
   ➥ background-color: #EEE;color:black;}❷
   li {display:inline;  border: solid 1px white;padding: 0 10px 0 10px;}❸
   img {float:right;}❹
   a{text-decoration: none;color: white;}
  </style>
 </head>
 <body>
  <div id="menu">❺
   <h1>   Référence HTML 5</h1>❻
   <ul>❼
    <li> <a  href="page1.html" tabindex="1" accesskey="A" title="Structure">
    ➥ Structure</a> </li>
    <li> <a  href="page2.html" tabindex="2" accesskey="B" title="Texte">Texte</a>
    ➥ </li>
    <!-- Suite de la liste -->
   </ul>
  </div>
  <div id="corps">❽
   <h1>La déclaration &lt;!DOCTYPE&gt;</h1>
   <img src="couvhtml5_gd.jpg" alt="HTML 5 et CSS 3" />
   <p>In principio creavit Deus caelum et terram terra autem erat inanis et vacua
   ➥ et tenebrae super faciem abyssi et spiritus Dei ferebatur super aquas dixitque
   ➥ Deus fiat lux et facta est lux et vidit Deus lucem quod esset bona et divis
   ➥ . . . </p>
  </div>
 </body>
</html>
```

La figure 12-11 montre le résultat obtenu.

Figure 12-11

Le positionnement en deux blocs horizontaux

Nous pouvons également envisager une variante de ce premier cas pour montrer que nous pouvons positionner des éléments de manière absolue à l'intérieur d'un élément lui-même positionné de cette façon. Nous réalisons cette opération à l'intérieur du second élément `<div>` de l'exemple 12-7 en conservant intégralement son code HTML 5.

Pour positionner les éléments enfants du second élément `<div>`, nous pouvons par exemple modifier simplement les styles CSS de la manière suivante :

```
<style type="text/css">
   body{font-size: 18px;}
   h1{font-size: 2em; margin-top: 5px;}
   div#menu {position: absolute; width:100%; height: 110px; left:0px; top:0;
   ➥ background-color:rgb(255,102,5);color:white; }
   div#corps { position: absolute; width:100%; left:0; top:110px; color:black;}❶
   li {display:inline; border: solid 1px white;padding: 0 10px 0 10px;}
   div#corps h1{position: absolute; width:600px; height:40px;top: 20px;left: 300px;
   ➥ background-color: #AAA;margin: 0;}❷
   img {position: absolute; top: 70px;right:20px;}❸
   p {position: absolute; width: auto; top: 70px; right:260px; left:30px; margin: 0;
   ➥ text-align: justify;background-color: #BBB;}❹
   a{text-decoration: none;color: white;}
</style>
```

Le second élément `<div>` garde son positionnement par rapport à la page (repère ❶), mais les éléments qu'il contient sont à leur tour positionnés. Ses éléments enfants `<h1>`, `<img />` et `<p>` étant eux-mêmes positionnés de manière absolue, ils ne le sont pas par rapport à la page mais par rapport à leur parent. L'élément `<h1>` a une largeur de 600 pixels et une hauteur de 40 pixels ; il est placé à 300 pixels du bord gauche de son parent et à 20 pixels de son bord supérieur (soit 110 + 20 = 130 pixels du bord supérieur de la page, repère ❷).

L'image est placée à 70 pixels du bord haut de son conteneur et à 20 pixels de son bord droit (repères ❷ et ❸). Elle n'est pas dimensionnée explicitement et conserve ses dimensions intrinsèques qui sont celles du fichier image. Le paragraphe `<p>` (repère ❹) n'est pas non plus dimensionné explicitement et sa largeur est fixée avec la valeur `auto`. Il sera ainsi redimensionné automatiquement sans empiéter sur l'image si la fenêtre est elle-même redimensionnée. En pratique, c'est son positionnement à 260 pixels du bord droit et à 30 pixels du bord gauche qui conditionne sa largeur.

Notons que si nous déplacions le conteneur de tous ces éléments en modifiant les propriétés `left` et `top` du second élément `<div>`, la position de ces trois éléments à l'intérieur de leur conteneur resterait inchangée. La figure 12-12 montre le résultat obtenu avec ces positionnements.

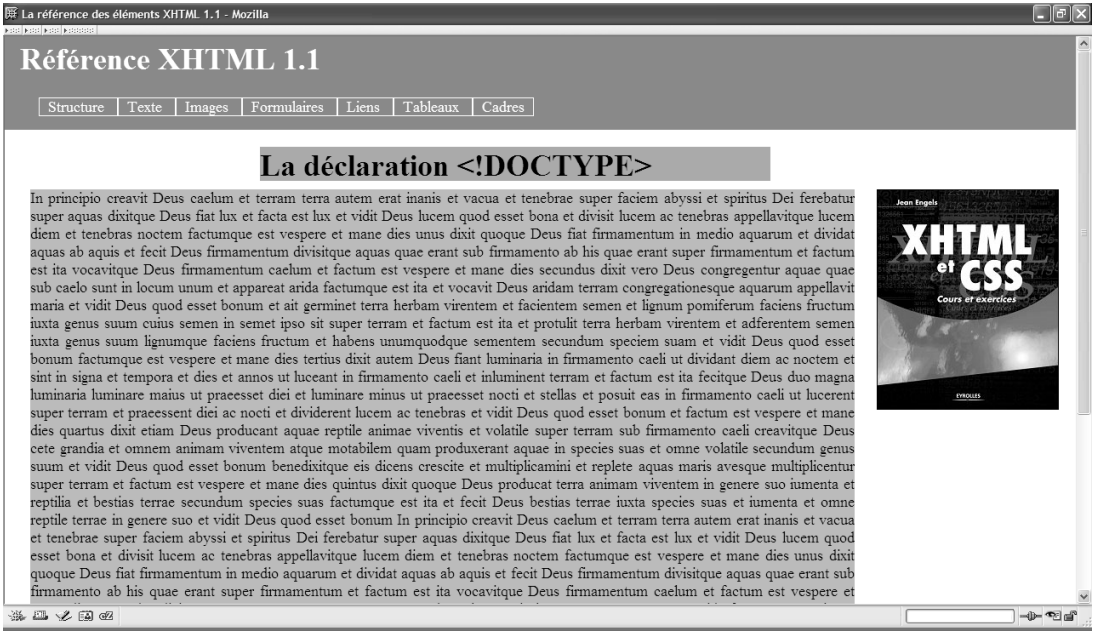


Figure 12-12

Positionnement dans un élément lui-même positionné

L'exemple suivant permet de créer un autre type de présentation très classique qui consiste à diviser la page en trois zones. La première est un bandeau qui peut contenir le titre du site ou une publicité, l'espace restant étant séparé en deux colonnes de la même façon que nous l'avons réalisé avec des cadres au chapitre 8. Le contenu est identique à celui des exemples précédents mais il est structuré différemment. À chaque zone correspond un élément `<div>` qui va être positionné de manière absolue dans la page. Nous n'avons pas défini ici d'attributs `id` pour chacun d'eux, mais nous appliquons des classes différentes à chaque division.

Le premier élément `<div>` (repère ❹) qui contient un titre `<h1>` (repère ❺) est positionné comme précédemment en haut de la page après avoir été dimensionné à 100 % en largeur et 100 pixels en hauteur. On évite ici de définir une hauteur en pourcentage pour qu'elle ne dépende pas du contenu de la page. Ces styles sont définis dans la classe `div.tete` (repère ❶).

Le reste de la page est partagé en deux colonnes. La colonne de gauche (repère ❻) contient le même menu (repère ❼) que dans l'exemple 12-7, mais dans un affichage vertical, soit son style par défaut. Cet élément `<div>` est d'abord dimensionné avec une largeur de 20 % de celle de la page et une hauteur de 100 %. Il est positionné de manière absolue au moyen de la classe `div.menu` à 100 pixels du bord haut de la page et à 0 pixel de son bord gauche (repère ❷).

Le contenu éditorial de la page constitue la colonne de droite créée avec le troisième élément `<div>` (repère ❽). Il contient un titre (repère ❾) et un paragraphe (repère ❿) comme dans les exemples précédents. Son traitement est assuré par la classe `div.contenu` dans laquelle il est dimensionné en largeur à 78 % et avec la valeur `auto` en hauteur. Son positionnement est absolu et il est placé à 20 % de la largeur de la page du bord gauche et à 100 pixels du bord supérieur (repère ❸). Avec leurs dimensions et positionnement respectifs, les trois zones sont bien collées les unes aux autres.

Exemple 12-8 Division de la page en trois zones

```
<!DOCTYPE html>
<html>
 <head>
  <meta http-equiv="Content-type" content="text/html;charset=UTF-8" />
  <title> La référence des éléments  HTML 5 </title>
   <style type="text/css">
   body{font-size: 18px;}
   div.tete {position: absolute ; left:0px;top:0px; width:100%; height:100px;
   ➥ background-color:rgb(0,0,153);margin:0; } ❶
   div.menu {position: absolute ; width:20%; height: 100%; left:0px; top:100px;
   ➥ background-color:rgb(255,102,51);color:white; } ❷
   div.contenu {margin: 1% ; position: absolute ;width:78%; left:20%;
   ➥ top:100px;background-color:white;color:black;} ❸
   div h1 {font-size:50px;font-style:italic;color:rgb(255,102,51);
   ➥ margin-top:0px;margin-left:200px;}
```

```
    li {padding: 15px;}
    p{text-align: justify;}
    img{float: left;margin: 0 10px 0 0;}
   </style>
  </head>
  <body>
   <div class="tete"> ❹
    <h1>   Référence HTML 5</h1> ❺
   </div>
   <div class="menu"> ❻
    <ul> ❼
     <li> <a  href="page1.html" tabindex="1" accesskey="A" title="Structure">
     ➥ Structure</a> </li>
     <li> <a  href="page2.html" tabindex="2" accesskey="B" title="Texte">Texte</a>
     ➥ </li>
     <li> <a  href="page3.html" tabindex="3" accesskey="C" title="Images">Images</a>
     ➥ </li>
     <li> <a  href="page4.html" tabindex="4" accesskey="D" title="Formulaires">
     ➥ Formulaires</a> </li>
     <li> <a  href="page5.html" tabindex="5" accesskey="E" title="Liens">Liens</a>
     ➥ </li>
     <li> <a  href="page6.html" tabindex="6" accesskey="G" title="Tableaux">
     ➥ Tableaux</a> </li>
     <li> <a  href="page7.html" tabindex="7" accesskey="H" title="Cadres">
     ➥ Cadres</a> </li>
    </ul>
   </div>
   <div class="contenu"> ❽
    <h1>La déclaration &lt;!DOCTYPE&gt;</h1> ❾
    <p>In principio creavit Deus caelum et terram terra autem erat inanis et vacua
    ➥ et tenebrae super faciem abyssi et spiritus Dei ferebatur super aquas dixitque
    ➥ Deus fiat lux et facta est lux et vidit Deus lucem quod esset bona et divisit
    ➥ lucem . . .</p> ❿
   </div>
    </body>
  </html>
```

La figure 12-13 montre le résultat obtenu avec ces définitions de styles. Nous pouvons remarquer une fois de plus l'intérêt des styles CSS car la présentation n'est pas la même que celle de la figure 12-12 alors que le contenu est le même.

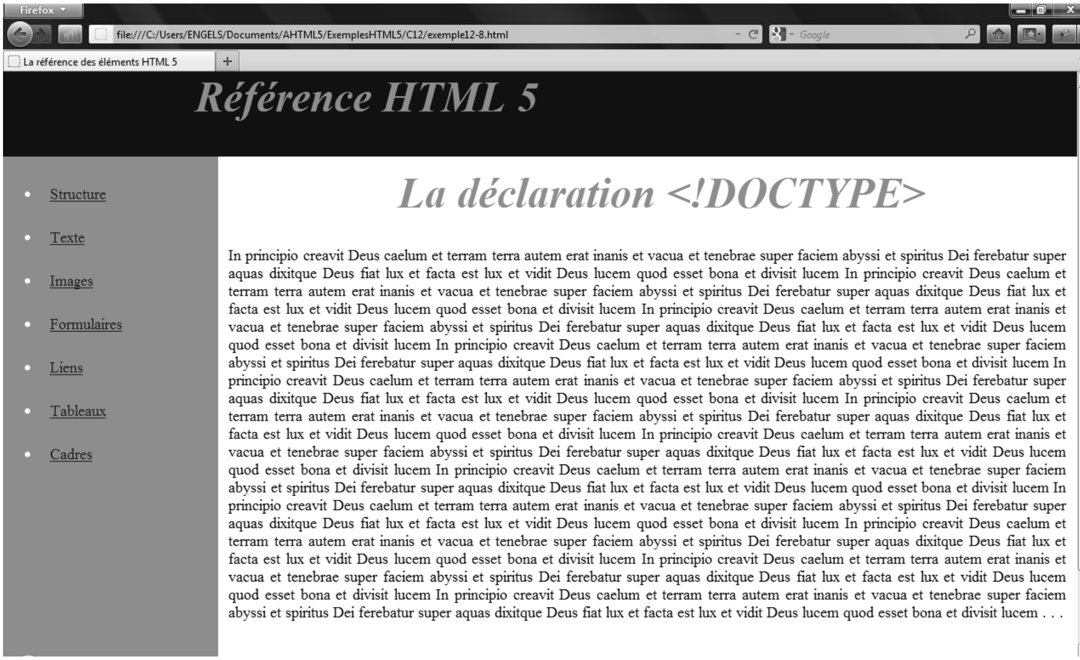


Figure 12-13

Positionnement en trois zones

Notre dernier exemple constitue une structure de page complexe comportant un bandeau, trois colonnes et un pied de page qui inclut l'adresse de contact. La première contient le menu, la deuxième un contenu textuel, et la troisième une liste de liens utiles. La figure 12-14 montre la présentation de la page que nous allons obtenir.

Chacune de ces zones est créée par un élément `<div>` dimensionné puis positionné de manière absolue en lui appliquant les classes `div.haut`, `div.gauche`, `div.droit`, `div.contenu` et `div.bas`. La seule division qui représente un élément nouveau par rapport aux cas antérieurs est celle qui permet de placer la division d'adresse en bas de la zone centrale de contenu. Cette division est incluse dans la division précédente et non plus directement dans `<body>`.

Dans le code de l'exemple 12-9, les éléments `<div>` sont volontairement placés dans le désordre afin de démontrer que leur position dans le code HTML 5 n'a aucune importance. Les définitions et les rôles des classes sont les suivants :

- `div.haut` (repères ❶ et ⓬) : largeur 100 %, hauteur 70 pixels, positionnée en haut et à gauche (`top:0` et `left:0`) ;
- `div.gauche` (repères ❷ et ⓫) : largeur 15 %, hauteur 100 % (ou `auto`), positionnée à 70 pixels du haut et à 0 du bord gauche ;
- `div.droit` (repères ❸ et ❼) : largeur 15 %, hauteur 100 % (ou `auto`), positionnée à 70 pixels du haut et à 0 du bord droit ;

- `div.contenu` (repères ❹ et ❽) : largeur 70 %, hauteur `auto` car la hauteur du paragraphe peut varier si on réduit la fenêtre du navigateur ;
- `div.bas` (repères ❺ et ❿) : elle s'applique à la division incluse dans le contenu. Sa largeur est de 100 % de celle de son parent (soit 70 % de celle de la page) et sa hauteur de 60 pixels. Elle est positionnée à gauche et en bas de son parent.

Afin que son contenu ne cache pas la fin du texte, on notera qu'il faut que le paragraphe (repère ❾) ait une marge basse d'au moins 60 pixels (repère ❻).

Figure 12-14

Organisation d'une page complexe en cinq zones

Exemple 12-9 Création d'une structure complexe en positionnement absolu

```
<!DOCTYPE html>
<html>
 <head>
  <meta http-equiv="Content-type" content="text/html;charset=UTF-8" />
  <title> La référence des éléments  HTML 5 </title>
  <style type="text/css">
   body{font-size: 18px;background-color:white;color:black;}
   div.haut {position: absolute ; left:0px;top:0px; width:100%; height:70px;
   ➥ background-color:rgb(0,0,153);margin:0; }❶
   div.gauche {position: absolute ; width:15%; height: auto; left:0px; top:70px;}❷
```

```
  div.droit{position: absolute ; width:15%; height: 100%; right:0; top:70px; }❸
  div.contenu {position: absolute ;width:70%; height:auto;left:15%; top:70px;
  ➥ padding: 0 10px 0 10px;border-left: 1px solid black; border-right: 1px solid
  ➥ black; }❹
  div.bas{position: absolute ; left:0px; bottom:0px; width:100%; height:60px;
  ➥ background-color:rgb(0,220,153);}❺
  div h1 {font-size:50px;font-style:italic;color:rgb(255,102,151);margin-top:0px;
  ➥ margin-left:200px;}
  li {padding: 15px;}
  p{text-align: justify;margin-bottom:60px;}❻
  img{float: left;margin: 0 10px 0 0;}
 </style>
</head>
<body>
 ❼<div class="droit">
   <ul>
     <li> <a  href="html10.html" tabindex="1" accesskey="A" title="Structure">
     ➥ XHTML 1.0</a> </li>
     <li> <a  href="page2.html" tabindex="2" accesskey="B" title="Texte">
     ➥ XHTML 1.1</a> </li>
     <li> <a  href="page3.html" tabindex="3" accesskey="C" title="Images">
     ➥ XHTML frameset</a> </li>
     <li> <a  href="page4.html" tabindex="4" accesskey="D" title="Formulaires">
     ➥ CSS 1</a> </li>
     <!—Suite de la liste -->
  </ul>
 </div>
 ❽<div class="contenu">
  <h1>La déclaration &lt;!DOCTYPE&gt;</h1>
  <img src="couvhtml5_gd.jpg" alt="HTML 5 et CSS 5" />
  ❾<p>In principio creavit Deus caelum et terram terra autem erat inanis et vacua
  ➥ et tenebrae super faciem abyssi et spiritus Dei ferebatur super aquas dixitque
  ➥ Deus fiat lux et facta est lux et vidit Deus lucem quod esset . . .</p>
 ❿<div class="bas">
  Contactez moi pour toutes questions sur HTML 5   <address>html@funhtml.com
  ➥ </address>
 </div>
 </div>
 ⓫<div class="gauche">
  <ul>
   <li> <a  href="page1.html" tabindex="1" accesskey="A" title="Structure">
   ➥ Structure</a> </li>
   <li> <a  href="page2.html" tabindex="2" accesskey="B" title="Texte">Texte</a>
   ➥ </li>
   <li> <a  href="page3.html" tabindex="3" accesskey="C" title="Images">Images</a>
   ➥ </li>
   <!—Suite de la liste -->
  </ul>
 </div>
```

```
⓬<div class="haut">
  <h1>   Référence HTML 5 </h1>
 </div>
</body>
</html>
```

Positionnement fixe

Le positionnement fixe n'est qu'un cas particulier du positionnement absolu. La détermination des dimensions et de la position d'un élément se réalisent de la même façon à l'aide des propriétés `width` et `height` pour les dimensions, et `top`, `left`, `right` et `bottom` pour la position. La seule différence réside dans le fait que le bloc conteneur utilisé n'est plus forcément l'élément parent mais la fenêtre du navigateur. De cette façon, un élément positionné de manière fixe reste à sa place, même en cas de défilement de la page quand la hauteur de celle-ci dépasse celle de la fenêtre du navigateur. En d'autres termes, les propriétés de positionnement sont appliquées non plus par rapport au conteneur mais par rapport aux bords de la fenêtre.

Ce type de positionnement peut être réalisé en utilisant encore la propriété `position` en lui donnant la valeur `fixed`. Elle est bien gérée par les navigateurs actuels comme Firefox.

Si nous reprenons exactement le même code HTML 5 que celui de l'exemple 12-9, nous pouvons réaliser un positionnement fixe des zones haute, gauche, droite et basse qui sont des parties de navigation qu'il peut être utile de voir en permanence (repères ❶, ❷, ❸ et ❺). Le contenu reste en positionnement absolu (repère ❹). Pour cela, nous écrivons les styles suivants.

Exemple 12-10 Le positionnement fixe

```
<style type="text/css">
   body{font-size: 18px;background-color:white;color:black;}
   div.haut {position: fixed ❶; left:0px;top:0px; width:100%; height:10%;
   ➥ background-color:rgb(0,0,153);margin:0; }
   div.gauche {position: fixed ❷; width:15%; height: auto; left:0px; top:10%;}
   div.droit{position: fixed ❸; width:15%; height: 100%; right:0; top:10%; }
   div.contenu {position: absolute ❹;width:70%; height:auto;left:15%;
   ➥ top:10%;padding: 0 10px 0 10px;border-left: 1px solid black;
   ➥ border-right: 1px solid black; }
   div.bas{position: fixed ❺; left:0px; bottom:0px; width:100%; height:60px;
   ➥ background-color:rgb(0,220,153);}
   div h1 {font-size:50px;font-style:italic;color:rgb(255,102,151);margin-top:0px;
   ➥ margin-left:200px;}
   li {padding: 15px;}
   p{text-align: justify;margin-bottom:60px;}
   img{float: left;margin: 0 10px 0 0;}
  </style>
```

Les seules modifications effectuées consistent à remplacer le mot-clé `absolute` par `fixed` dans toutes les classes sauf celle du contenu qui doit pouvoir défiler pour être lisible. La figure 12-15 montre le résultat obtenu dans une fenêtre redimensionnée et après avoir effectué un défilement vertical. Nous y constatons que seule la zone centrale a défilé alors que les autres n'ont pas bougé et restent entièrement visibles. Nous pouvons également remarquer que la zone basse est maintenant bien positionnée par rapport à la fenêtre car elle occupe toute la largeur de l'écran, contrairement au cas précédent de positionnement absolu.

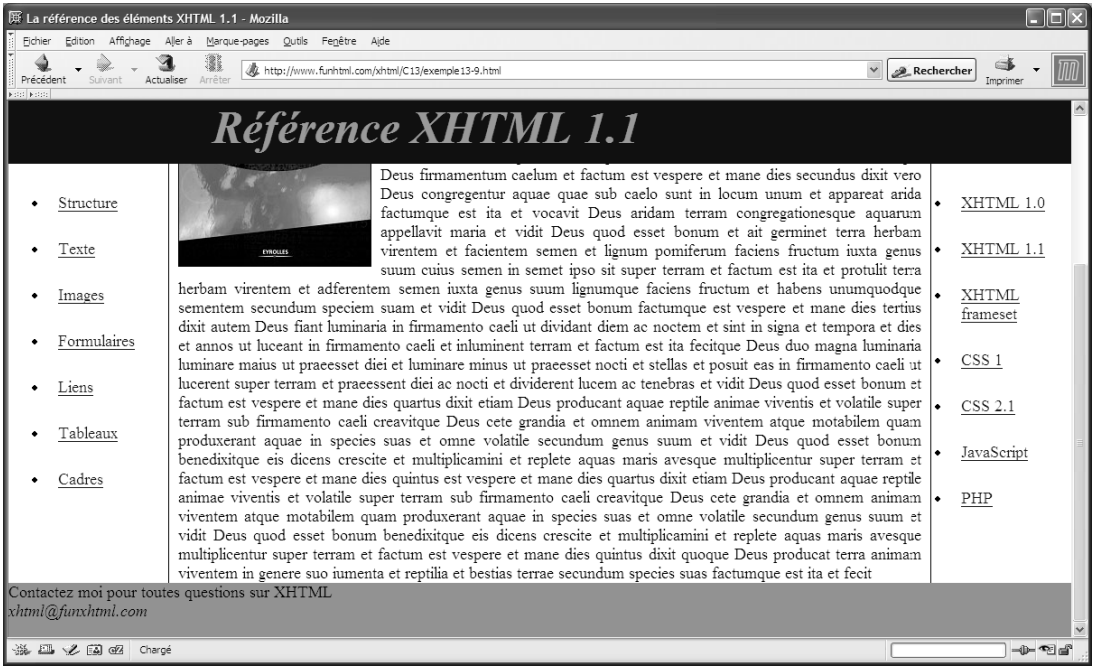


Figure 12-15

Le positionnement fixe

Le positionnement fixe représente donc une alternative crédible à la création de cadres. Nous parlerons plutôt de similicadres qui donnent l'illusion d'une page avec cadres et en présente les avantages visuels, mais sans interactivité entre les différentes zones comme c'est le cas des cadres. La procédure à suivre, relativement simple, est la suivante :

- créer autant d'éléments `<div>` que l'on désire obtenir de zones différentes dans la page. Chaque zone est l'équivalent des éléments `<frame />` utilisés dans la méthode avec cadres. Pour chacune de ces zones, définir la propriété `position` avec la valeur `fixed` ;
- dimensionner chacune de ces divisions en utilisant les propriétés `width` (largeur) et `heigth` (hauteur) ;

- positionner les éléments `<div>` en définissant les propriétés `left`, `top`, `right` et `bottom` qui nous permettent de placer les éléments `<div>` par rapport aux bords de la fenêtre ;
- définir éventuellement les propriétés de couleur de fond, de bordure, de marge et d'espacement pour améliorer la présentation du contenu de chaque élément `<div>`.

Pour retrouver une interactivité entre les similicadres, par exemple afin que le clic sur un lien du menu affiche un contenu adapté dans la zone centrale, nous pouvons créer plusieurs pages qui ont toutes en commun les divisions positionnées de manière fixe et dont le contenu de la zone centrale est variable. L'illusion d'un site avec cadres est alors complète.

Visibilité et ordre d'empilement

Nous avons pu constater qu'en positionnant des éléments de manière relative, absolue ou fixe, il était possible que plusieurs éléments occupent partiellement ou totalement le même espace dans la fenêtre du navigateur. Dans ce cas, le dernier apparu dans l'ordre du code HTML 5 se superpose au précédent. Pour pouvoir intervenir sur cet état de fait et gérer volontairement ces superpositions, CSS définit un placement des éléments selon trois dimensions, les deux premières dans le plan de l'écran, sur lesquelles nous pouvons intervenir avec les propriétés `left`, `top`, `right` et `bottom` comme nous l'avons déjà vu, et la troisième selon un « axe des z » perpendiculaire à l'écran et dirigé vers le spectateur. Nous pouvons gérer cet ordre d'empilement au moyen de la propriété `z-index` dont la syntaxe est la suivante :

```
z-index : auto | <Nombre> | inherit
```

Elle ne s'applique qu'aux éléments positionnés, et les valeurs qu'elle peut prendre sont les suivantes.

- `auto` : l'élément a la même valeur que celle de son parent direct, qu'il soit défini explicitement par un nombre ou implicitement par son ordre d'apparition dans le code HTML 5 (le dernier ayant la priorité).
- `<Nombre>` : un nombre entier positif ou négatif, sachant que plus le nombre est grand, plus l'élément est placé en avant, et se superpose à ceux dont la valeur est inférieure.

Nous pouvons intervenir dynamiquement sur l'ordre d'empilement au moyen de code JavaScript en modifiant la valeur de la propriété CSS `z-index` (qui en JavaScript porte le nom de `zIndex`) en réponse à une action du visiteur (survol de l'élément par la souris, clic…). Cette modification est gérée par les attributs gestionnaires d'événements correspondants (`onmouseover`, `onclick`…) ou par la pseudo-classe `:hover` par exemple.

Dans le même ordre d'idées, nous pouvons appliquer à tous les éléments la propriété `visibility` qui permet de les cacher ou de les rendre visibles. Sa syntaxe est la suivante :

```
visibility : visible | hidden | collapse | inherit
```

- `visible` : l'élément est visible normalement et c'est la valeur par défaut.

- `hidden` : l'élément est caché mais la différence de comportement avec la propriété `display`, quand elle prend la valeur `none`, est que la boîte de l'élément est ici maintenue dans la page mais qu'elle est simplement vide et non retirée de la page comme avec `display`.
- `collapse` : son comportement est identique à la valeur `hidden` mais elle s'applique particulièrement aux cellules des tableaux.

Cette propriété est héritée par défaut et elle s'applique donc aux éléments enfants.

L'exemple 12-11 donne une illustration de la gestion de l'ordre d'empilement et de la visibilité de plusieurs éléments. La page comporte une division (repère ❻) qui contient un paragraphe `<p>` positionné (repères ❺ et ❼), lui-même incluant du texte et une image flottante (repères ❽ et ❸). La division comprend également deux images (repères ❾ et ❿) qui y sont positionnées de manière absolue (repères ❶ et ❷). La figure 12-16 montre le résultat obtenu initialement lors de l'affichage de la page.

Figure 12-16

L'état initial de la page avec des éléments empilés

Compte tenu de l'ordre d'apparition des éléments enfants de la division `<div>`, l'ordre d'empilement sans utilisation de la propriété `z-index` devrait être de l'arrière vers l'avant, soit le paragraphe, la première image puis la seconde au premier plan. Cependant, comme nous attribuons à la propriété `z-index` de la première image la valeur `2` et à la suivante la valeur `1`, cet ordre est inversé.

Si la propriété `z-index` ne servait qu'à définir un ordre d'empilement fixe, elle n'aurait qu'un intérêt limité car il suffirait de placer en dernier dans le code HTML 5 l'élément que l'on veut voir se positionner au premier plan. En gérant l'événement `onclick`, par exemple pour le paragraphe ou chacune des images avec le code JavaScript suivant :

```
onclick="this.style.zIndex++"
```

nous permettons au visiteur de placer au premier plan l'élément qu'il désire et éventuellement d'inverser cet ordre à chaque nouveau clic. Notons de nouveau qu'en JavaScript le nom de la propriété devient `zIndex` et que l'opérateur ++ signifie simplement qu'il faut augmenter la valeur de la propriété de 1 unité. De même, le texte du paragraphe, recouvert partiellement par la première image, peut être mis au premier plan pour être entièrement lisible.

La visibilité de l'image incluse dans le paragraphe (repère ❽) est contrôlée par la pseudo-classe `:hover` (repère ❹). Si le visiteur positionne le curseur sur cette image (en le laissant immobile, sinon on obtient un effet de clignotement), elle devient invisible en laissant l'espace qu'elle occupait vide dans le paragraphe. Cet effet est obtenu en donnant à la propriété `visibility` la valeur `hidden` en cas de survol (il est annulé automatiquement quand le curseur quitte la zone de l'image).

Exemple 12-11 Visibilité et empilement

```
<!DOCTYPE html>
<html>
 <head>
  <meta http-equiv="Content-type" content="text/html;charset=UTF-8" />
  ➥ <title>Visibilité et ordre d'empilement</title>
  <link rel="shortcut icon" type="images/x-icon" href="../images/favicon.ico" />
  <style type="text/css" >
   body{font-size:18px;}
   img.un{position:absolute;top:160px;left:350px;}❶
   img.deux{position:absolute;top:260px;left:450px;}❷
   img#couv{float: left;}❸
   img#couv:hover{visibility:hidden;}❹
   p{background-color: #EEE; position:absolute; top:40px;  left:50px;}❺
  </style>
 </head>
 <body>
  ❻<div>
   ❼<p onclick="this.style.zIndex++"><b>HTML 5 </b>:
   ❽<img id="couv" src="couvhtml5_gd.jpg" alt="HTML 5 et CSS 3" height="130"
   ➥ width="107" />
    In principio creavit Deus caelum et terram terra autem erat inanis et vacua
    ➥ et tenebrae super faciem abyssi et spiritus Dei ferebatur super aquas
    ➥ dixitque Deus fiat lux et facta est lux et vidit Deus lucem quod esset bona
    ➥ et divisit lucem ac tenebras appellavitque lucem diem et tenebras noctem
    ➥ factumque est vespere et mane dies unus dixit </p>
   ❾<img src="couvhtml5_gd.jpg" alt="HTML 5 et CSS 3" height="285" width="240"
   ➥ class="un" onclick="this.style.zIndex++" />
```

```
    ❿<img src="couvphp5gd.png" alt="PHP 5" height="285" width="240" class="deux"
    ➥ onclick="this.style.zIndex++" />
  </div>
 </body>
</html>
```

La figure 12-17 donne le résultat obtenu après avoir successivement placé le paragraphe devant la première image (en cliquant sur le paragraphe), la seconde image devant la première (en cliquant sur la seconde), puis en plaçant le curseur sur l'image incluse dans le paragraphe. Par une série de nouveaux clics, il est possible de retrouver l'état initial.

Figure 12-17

L'état de la page après la modification de l'empilement et de la visibilité de certains éléments

Affichage multicolonne en CSS 3

Afficher un texte ou d'autres contenus sur plusieurs colonnes était possible en multipliant les éléments `<div>` et en les positionnant comme nous l'avons exposé plus haut. Désormais, avec CSS 3, il est très facile d'afficher ces contenus sur plusieurs colonnes grâce à plusieurs propriétés dont les principales sont `column-count` et `column-width` dont les syntaxes sont :

```
column-count: N | auto;
column-width: Largeur | auto;
```

Dans la première, le paramètre `N` désigne le nombre de colonnes ; dans la seconde, il faut indiquer la largeur de chaque colonne en pixels ou en nombre de caractères. Dans ce cas, le nombre de colonnes n'est pas fixe et dépend de la taille de la fenêtre du navigateur. On peut utiliser ces deux propriétés indépendamment ou ensemble ; dans ce dernier cas, le paramètre `auto` est appliqué quand une seule des deux propriétés a une valeur précisée. Il existe aussi un raccourci pour définir ces deux propriétés à la fois, la propriété :

```
columns: <largeur|auto> || <N | auto>
```

dont les valeurs sont une définition de largeur et un nombre de colonnes. Notons qu'il est possible de définir des colonnes pour un élément qui est inclus dans un autre élément lui-même divisé en plusieurs colonnes (voir l'exemple 12-12).

La présentation peut ensuite être améliorée avec d'autres propriétés. Pour régler l'espace entre les colonnes, on peut utiliser la propriété :

```
column-gap : N | normal;
```

dans laquelle `N` est un nombre exprimé en pixels ou en em par exemple.

Pour créer une ligne de séparation visuelle verticale entre les colonnes nous disposons des propriétés suivantes :

```
column-rule-width: N;
column-rule-style: <style border>;
column-rule-color: couleur;
```

qui définissent respectivement la largeur de la séparation, son style avec les mêmes valeurs possibles que pour une bordure et enfin la couleur désirée. Ces trois propriétés peuvent être définies en une seule fois avec le raccourci suivant :

```
column-rule: N  <style border>  couleur;
```

L'exemple 12-12 illustre ce que l'affichage multicolonne permet de réaliser. Tout d'abord, nous y définissons, pour les éléments `<div>` et `<p>`, le nombre de colonnes pour différents navigateurs (repère ❶), l'espace entre les colonnes (repère ❷), la largeur, le style puis la couleur de la ligne de séparation (repère ❹). Le code du repère ❸ illustre comment on peut définir chacun des éléments individuellement, cette définition spécifique pouvant être utilisée pour modifier dynamiquement une seule des caractéristiques. Le texte est ensuite justifié et se voit affecté une couleur de fond pour mieux visualiser le résultat obtenu dans la page (repère ❺). L'élément parent de tout le contenu est `<div>` (repère ❼) et il contient aussi bien un titre `<h1>` (repère ❽), des images (repères ❾ et ⓫) et un paragraphe (repère ❿), qui sera aussi divisé en trois colonnes. La seconde image (repère ⓫) utilise la classe `.flotte` (repère ❻) qui permet d'apprécier ce qu'il est possible d'afficher même à l'intérieur d'une colonne.

La figure 12-18 montre le résultat obtenu. On y remarque que si le contenu de la division est affiché sur trois colonnes, le paragraphe situé dans la première colonne est aussi divisé en trois. Ceci ouvre la porte à des mises en page très variées.

Exemple 12-12 Affichage multicolonne

```
<!DOCTYPE html>
<html>
 <head>
  <title>Multicolonne CSS 3</title>
  <meta charset="UTF-8" />
   <style type="text/css">
   p,div {
    /*  ❶  */
   -moz-column-count: 3;
   -webkit-column-count: 3;
   column-count: 3;
    /*  ❷  */
   -moz-column-gap:40px;
   -webkit-column-gap:40px;
    column-gap:40px;
    /*  ❸ */
    column-rule-color: #3366FF;
    column-rule-style: double;
    column-rule-width: 10px;
     /*  ❹  */
    -moz-column-rule: 10px double #3366FF;
    -webkit-column-rule: 10px double #3366FF;
    column-rule: 10px double #3366FF;
    /*  ❺  */
    text-align:justify;
    background-color: #EEE;
    }
    .flotte{display:block;float:right;padding:1em; width:158px;height:157px;} ❻
  </style>
 </head>
 <body>
  <div>❼
   <h1>In principio creavit Deus caelum et terram </h1>❽
   <img src="horloge.jpg" alt="heure"/>❾
   In principio creavit Deus caelum et terram terra autem erat inanis et  vacua et
   ➥ tenebrae super faciem abyssi et spiritus Dei ferebatur super aquas dixitque
   ➥ Deus fiat lux et facta est lux et vidit Deus lucem quod …

   <p>Deus aridam terram congregationesque aquarum appellavit maria et vidit Deus
   ➥ quod esset bonum et ait germinet terra herbam virentem et facientem semen et
   ➥ lignum pomiferum faciens fructum iuxta genus suum cuius…
   </p>❿
   Deus aridam terram congregationesque aquarum appellavit maria et vidit Deus quod
   ➥ esset bonum et ait germinet terra herbam virentem

   <img src="horloge.jpg" alt="heure" class="flotte" /> ⓫
```

```
et volatile super terram sub firmamento caeli creavitque Deus cete grandia et omnem
➥ animam viventem atque motabilem quam produxerant aquae in species suas et omne
➥ volatile secundum genus suum et vidit Deus quod esset …
  </div>
 </body>
</html>
```

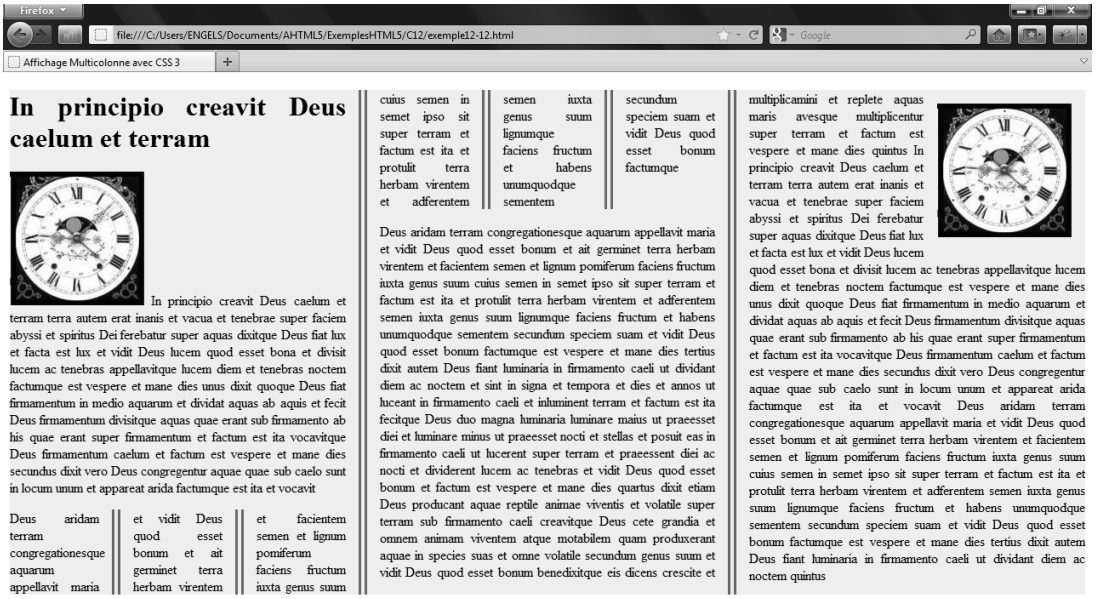


Figure 12-18

Affichage multicolonne de texte

Redimensionnement par l'utilisateur en CSS 3

Les propriétés que nous avons exposées dans ce chapitre permettent au webmaster de dimensionner des éléments, mais le visiteur du site ne pouvait pas jusqu'à présent agir sur la dimension de ceux-ci, si ce n'était en réduisant ou augmentant la taille de son navigateur. Ceci est corrigé avec l'apparition dans CSS 3 de la propriété `resize` qui permet au client de redimensionner à son goût et sans condition un ou plusieurs éléments présents dans une page. Je pense que cette propriété intéressante ne manquera pas d'être très utilisée, surtout quand elle sera fonctionnelle dans tous les navigateurs ; pour l'instant, c'est le cas avec Firefox, Chrome et Safari. Sa syntaxe est la suivante :

```
resize: horizontal | vertical | both;
```

Le redimensionnement est donc autorisé dans le sens horizontal, vertical ou les deux à la fois. Dans l'exemple 12-13 qui utilise cette propriété, nous créons trois paragraphes qui contiennent un texte et une image chacun (repères ❸, ❹ et ❺). Les styles définis pour ces paragraphes leur donnent des dimensions, hauteur et largeur (repère ❶), qui permettent de les voir tous sur une même ligne à l'ouverture de la page (voir la figure 12-19). Les paragraphes sont ensuite déclarés comme redimensionnables dans les deux sens (repère ❷). Cette propriété est donc très simple d'emploi. Sur la figure 12-20, on constate le résultat obtenu après redimensionnement des premier et troisième éléments, le premier étant agrandi pour pouvoir lire le texte en entier et le troisième réduit au maximum. L'utilisateur peut ensuite retrouver l'état initial soit en modifiant la taille des éléments, soit en rechargeant la page avec la touche F5 par exemple.

Exemple 12-13 Redimensionnement par l'utilisateur

```
<!DOCTYPE html>
<html>
 <head>
  <meta http-equiv="Content-type" content="text/html;charset=UTF-8" />
  <title>Redimensionnement par l'utilisateur'</title>
  <style type="text/css">
   p{
    height:190px;width:400px;❶
    resize:both;❷
    background-color:#DDD;overflow:hidden;
    display:inline-block;
    }
   img{
    float:right;padding:1em;
    }
   span{
    font-size: larger;background-color:#000;color:white;;
    }
  </style>
 </head>
 <body>
  <p>❸
   <span>Article Un</span>  <br />
   <img src="horloge.jpg" alt="heure"  />
   In principio creavit Deus caelum et terram terra autem erat inanis et vacua et
  ➥ tenebrae super faciem abyssi et spiritus Dei ferebatur super...
  </p>
  <p>❹
   <span>Article Deux</span> <br />
   <img src="horloge.jpg" alt="heure" />In principio creavit Deus caelum et terram
   ➥ terra autem erat inanis et vacua et tenebrae super faciem abyssi et spiritus
   ➥ Dei ferebatur super...
  </p>
  <p>❺
   <span>Article Trois</span> <br />
```

```
    <img src="horloge.jpg" alt="heure"/>In principio creavit Deus caelum et terram
    ➥ terra autem erat inanis et vacua et tenebrae super faciem abyssi et spiritus
    ➥ Dei ferebatur super...
   </p>
  </body>
 </html>
```

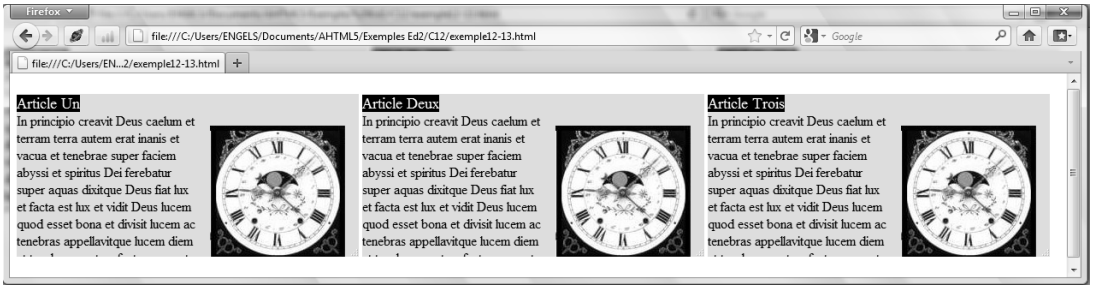


Figure 12-19

Trois éléments redimensionnables

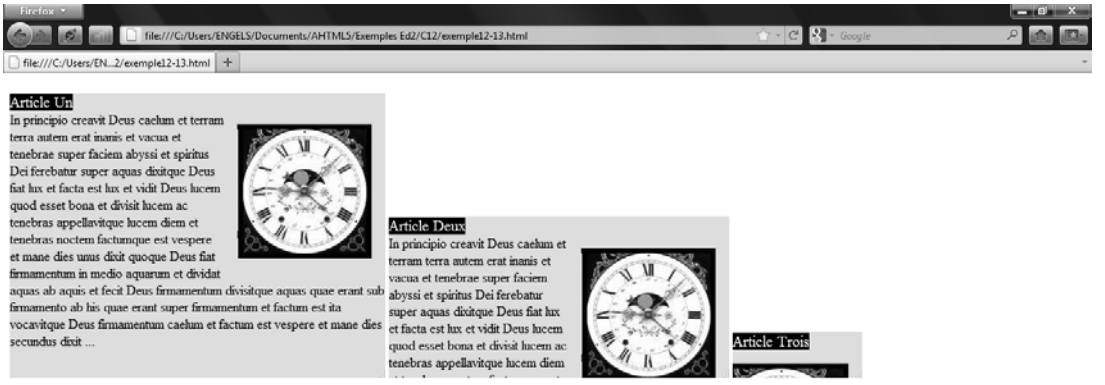


Figure 12-20

Les mêmes éléments redimensionnés par le visiteur

Transformations des éléments en CSS 3

Toutes les propriétés que nous allons étudier ici, apparues dans CSS 3, peuvent être considérées d'un nouveau type. Elles permettent d'effectuer des transformations géométriques sur des éléments, ou plus précisément sur leur conteneur, leur contenu ne faisant que suivre. Elles relèvent bien de transformations géométriques au sens mathématique du terme car il s'agit de translation, de rotation d'agrandissement ou de réduction (donc

d'homothétie) et enfin de déformation. La propriété fondamentale de ces opérations est `transform`, qui est utilisée dans tous les cas ; ce sont les valeurs qu'on lui donne qui déterminent le choix de la transformation.

Les translations en CSS 3

Une translation consiste à déplacer un objet en conservant sa forme et ses dimensions. La syntaxe à utiliser est :

```
transform:translate(X, Y);
```

dans laquelle `X` et `Y` sont les valeurs des déplacements horizontal et vertical en pixels, vers la droite et le bas si elles sont positives et l'inverse sinon. Ceci est réalisable dans tous les navigateurs avec les préfixes correspondants (`-moz-`, `-o-`, `-webkit-` et `-ms-`).

Dans l'exemple 12-14 nous plaçons deux images dans une page (repères ❻ et ❼) dont l'une a un identifiant qui nous permet de lui attribuer un style particulier. Ce dernier consiste en un déplacement de 550 pixels vers la droite et de 330 pixels vers le bas (repères ❶ à ❺), quand le curseur survole l'image, ceci à l'aide de la pseudo-classe `:hover`.

Exemple 12-14 Translation d'une image

```
<!DOCTYPE html>
<html>
 <head>
  <meta http-equiv="Content-type" content="text/html;charset=UTF-8" />
  <title>Translation d'une image'</title>
  <style type="text/css">
   #france:hover{
    -moz-transform: translate(550px,330px);❶
    -o-transform:translate(550px,330px);❷
    -webkit-transform:translate(550px,330px);❸
    -ms-transform:translate(550px,330px);❹
    transform:translate(550px,330px);❺
   }
  </style>
 </head>
<body>
   <img src="france.gif" id="france" alt="france"/> ❻
   <img src="france.gif" alt="france"/> ❼
 </body>
</html>
```

La figure 12-21 montre que la première image a été déplacée alors que la seconde est restée à sa place initiale.

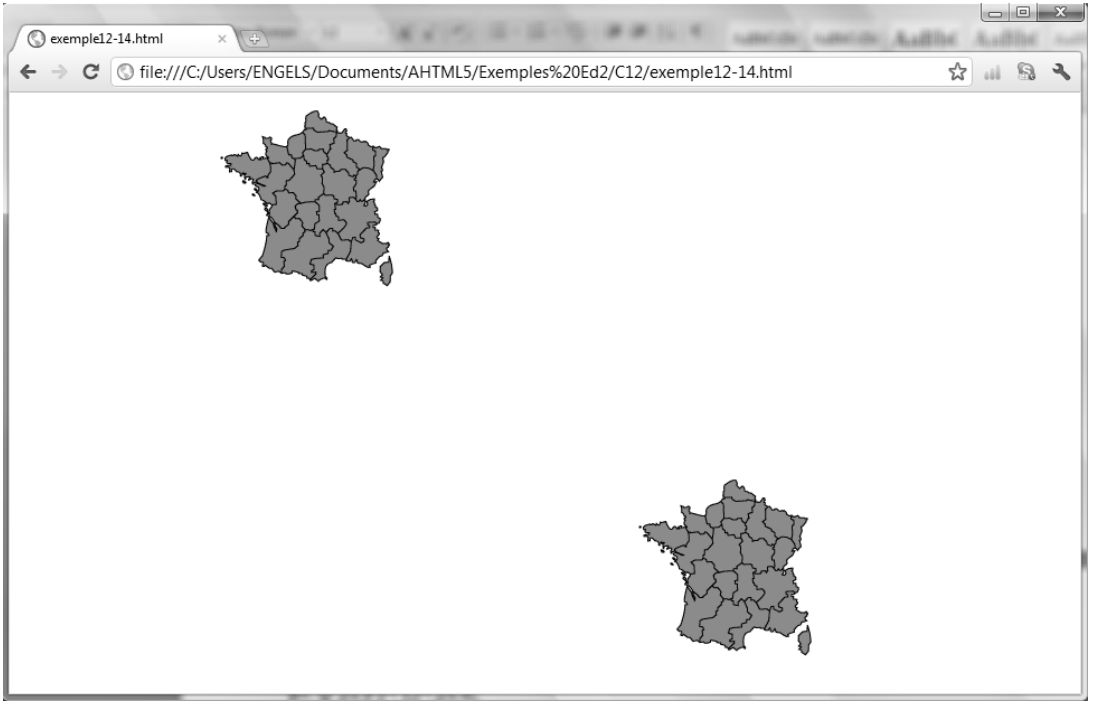


Figure 12-21

Translation d'une image

Les rotations en CSS 3

Pour définir une rotation, il faut préciser son angle et son centre. La définition de l'angle s'effectue avec la propriété `transform` et la fonction `rotate()` dont le paramètre est l'angle en degrés selon la syntaxe suivante :

```
transform:rotate(X deg);
```

Si `X` est positif la rotation est effectuée dans le sens horaire, sinon dans le sens trigonométrique. Par défaut, le centre de rotation est le centre de symétrie du conteneur de l'élément. Pour définir explicitement un centre de rotation différent, il faut utiliser la propriété `transform-origin` dont la syntaxe est :

```
transform-origin: X px Y px;
```

Le couple `X`, `Y` représentent les coordonnées du centre de rotation ; elles peuvent aussi être formulées à l'aide d'une combinaison des mots-clés `left`, `center` ou `right` pour `X` et `top`, `center` ou `bottom` pour `Y`. On obtient donc ainsi neuf possibilités élémentaires. Pour une définition plus précise, on peut définir `X` et `Y` en pourcentage des dimensions du conteneur,

sachant que l'origine des mesures est son coin supérieur gauche. Dans l'exemple 12-15, la page contient une division et un paragraphe de mêmes contenus (repères ⓫ et ⓬), pour mieux visualiser le résultat obtenu. En utilisant la pseudo-classe `:hover`, nous présentons un cas de rotation de 90 degrés dans le sens trigonométrique pour l'élément `<p>` (repères ❶ à ❺) quand il est survolé par le curseur. Le centre de rotation est défini comme étant le coin en bas et à droite du rectangle conteneur du paragraphe (repères ❻ à ❿). La figure 12-22 montre le résultat obtenu.

Exemple 12-15 Rotation d'un élément

```
<!DOCTYPE html>
<html>
 <head>
  <meta http-equiv="Content-type" content="text/html;charset=UTF-8" />
  ➥ <title>Rotation d'un élément</title>
  <style type="text/css">
   p,div{
    height:auto;width:300px;
    background-color:#EEE;
    margin:20px;
    display:inline-block;
    }
   p:hover{
    -moz-transform:rotate(-90deg);❶-moz-transform-origin: right bottom;❻
    -o-transform:rotate(-90deg);❷-o-transform-origin: right bottom;❼
    -webkit-transform:rotate(-90deg);❸-webkit-transform-origin: right bottom;❽
    -ms-transform:rotate(-90deg);❹-ms-transform-origin: right bottom;❾
    transform:rotate(-90deg);❺transform-origin: right bottom;❿
   }
  </style>
 </head>
 <body>
  ⓫<div>In principio creavit Deus caelum et terram terra autem erat inanis et
  ➥ vacua et tenebrae super faciem abyssi et spiritus Dei ferebatur super aquas
  ➥ dixitque Deus fiat lux et facta est lux et vidit Deus lucem quod esset bona
  ➥ et divisit lucem ac tenebras appellavitque lucem diem et tenebras noctem
  ➥ factumque est vespere et mane dies unus dixit..<br /><img src="horloge.jpg" />
  ➥ </div>
  ⓬<p>In principio creavit Deus caelum et terram terra autem erat inanis et
  ➥ vacua et tenebrae super faciem abyssi et spiritus Dei ferebatur super aquas
  ➥ dixitque Deus fiat lux et facta est lux et vidit Deus lucem quod esset bona
  ➥ et divisit lucem ac tenebras appellavitque lucem diem et tenebras noctem
  ➥ factumque est vespere et mane dies unus dixit..<br /><img src="horloge.jpg" />
  ➥ </p>
 </body>
</html>
```

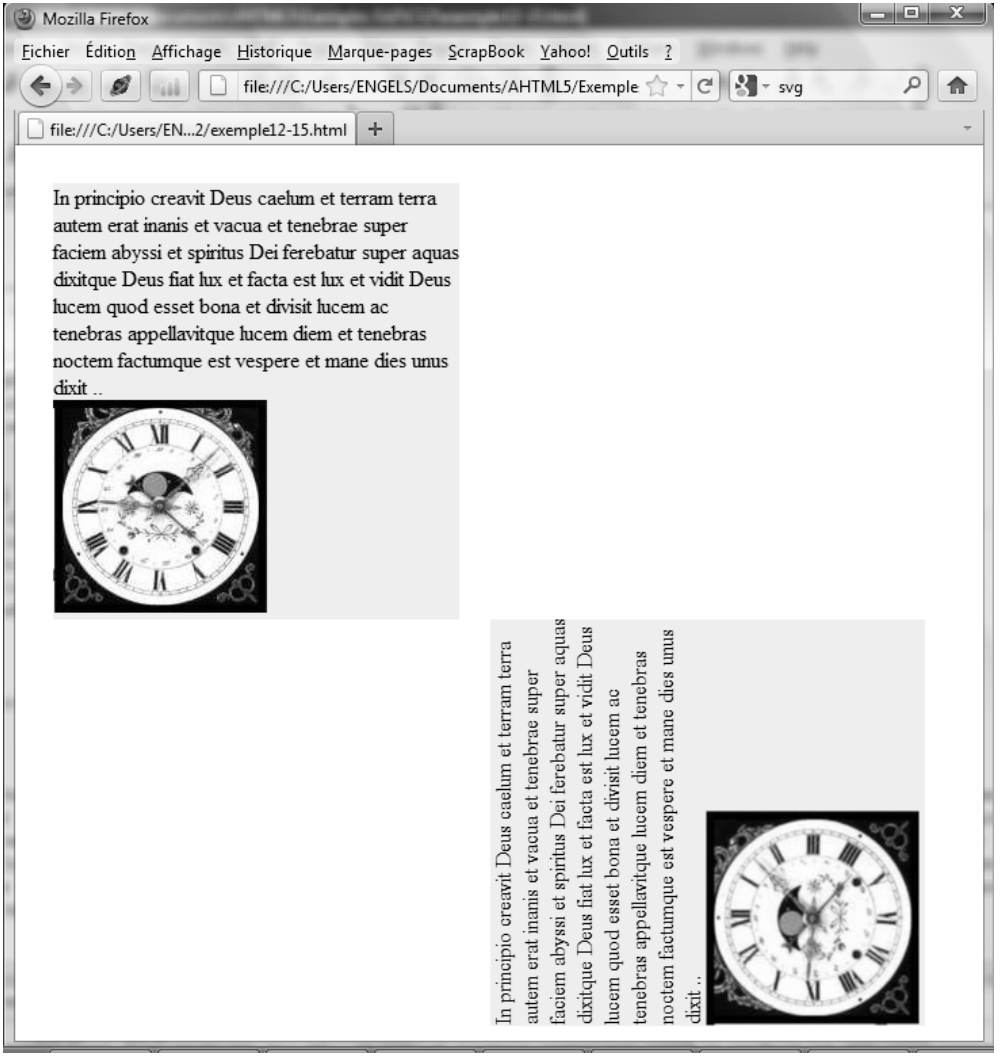


Figure 12-22

Rotation d'une image

Agrandissement et réduction en CSS 3

Nous avons déjà présenté la propriété `resize` qui permet à l'utilisateur de redimensionner un élément lui-même. Ici il s'agit plutôt d'effectuer le changement de dimensions à partir du code, par exemple en réponse à une action du visiteur, et éventuellement par du code JavaScript. Là encore nous utilisons la propriété `transform` avec la syntaxe :

```
transform:scale(X,Y);
```

dans laquelle `X` et `Y` sont les facteurs d'agrandissement (pour les valeurs supérieures à 1) ou de réduction (pour les valeurs inférieures à 1) respectivement horizontalement et verticalement. Dans le cas d'une image, celle-ci est déformée si `X` est différent de `Y`. Le passage d'un seul paramètre implique l'égalité des deux.

Dans l'exemple 12-16, nous plaçons deux images (repères ❿ et ⓫) dont la première est de taille réduite, par l'affectation de valeurs aux propriétés `width` et `height` (repère ❶). Lors du passage du curseur sur celle-ci, elle sera agrandie d'un facteur 10 grâce à la définition du style `scale(10)` (repères ❷ à ❺) ; la seconde sera réduite de 50 % horizontalement seulement avec `scale(0.5,1)` (repères ❻ à ❾). Les figures 12-23 et 12-24 montrent respectivement l'état initial et l'état final lors du passage du curseur sur la première image. Les applications éventuelles sont faciles à imaginer sur un site.

Exemple 12-16 Agrandissement et réduction d'éléments

```
<!DOCTYPE html>
<html>
 <head>
  <meta http-equiv="Content-type" content="text/html;charset=UTF-8" />
  <title>Agrandissement Réduction</title>
  <style type="text/css">
    #reduite{ width:60px;height:40px;}❶
    img:hover{
    -moz-transform:scale(10);❷
    -o-transform:scale(10);❸
    -webkit-transform:scale(10);❹
    -ms-transform:scale(10);❺
    margin-left:300px;  margin-top:150px;
    }
    #modif:hover{
    -moz-transform:scale(.5,1);❻
    -o-transform:scale(.5,1);❼
    -webkit-transform:scale(.5,1);❽
    -ms-transform:scale(.5,1);❾
    margin-left:120px;
    }
  </style>
 </head>
 <body>
  <img src="romy2011.jpg" id="reduite" alt="Romy"/>❿   <br />
  <img src="horloge.jpg" id="modif" alt="heure"/>⓫
 </body>
</html>
```

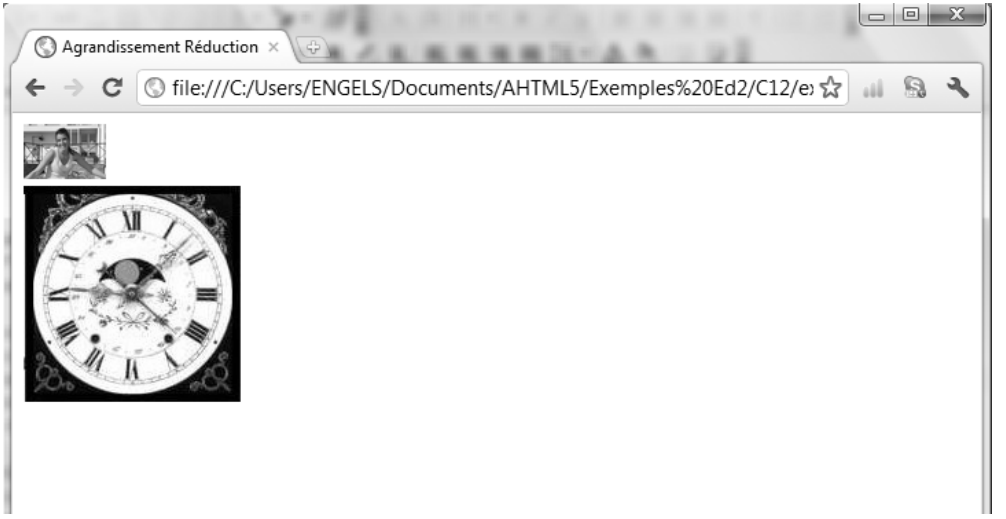


Figure 12-23

Agrandissement d'une image : état initial

Figure 12-24

Agrandissement d'une image : état final

Déformation de la boîte du conteneur

Les diverses transformations étudiées jusqu'à présent préservent sinon la taille du moins la forme de la boîte rectangulaire de l'élément. La fonction `skew()`, que l'on peut affecter à la propriété `transform`, permet d'effectuer une déformation de celle-ci par la combinaison de deux rotations appliquées cette fois, non plus à la boîte entière, mais à ses côtés horizontaux et verticaux formant ainsi un parallélogramme, et déformant son contenu. Sa syntaxe, qui doit être augmentée des préfixes habituels, est :

```
transform:skew(Xdeg, Ydeg);
```

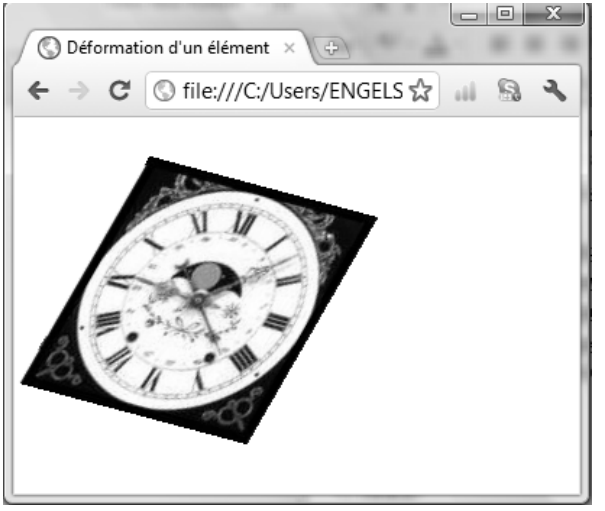
où `X` est l'angle de rotation des côtés verticaux et `Y` celui des côtés horizontaux exprimés en degrés, les valeurs pouvant être négatives.

Dans l'exemple 12-17, nous incluons une image (repère ❻) qui va subir une déformation par rotation des côtés verticaux de 30 degrés et des côtés horizontaux de 15 degrés, grâce à l'application du style (repères ❶ à ❺) quand le curseur la survole. La figure 12-25 montre le résultat obtenu.

Exemple 12-17 Déformation d'une image

```
<!DOCTYPE html>
<html>
 <head>
  <meta http-equiv="Content-type" content="text/html;charset=UTF-8" />
  <title>Déformation d'un élément</title>
  <style type="text/css">
   #img1:hover{
    -moz-transform:skew(30deg,15deg);❶
    -o-transform:skew(30deg,15deg);❷
    -webkit-transform:skew(30deg,15deg);❸
    -ms-transform:skew(30deg,15deg);❹
    transform:skew(30deg,15deg);❺
    margin:40px;
   }
  </style>
 </head>
 <body>
   <img id="img1" src="horloge.jpg" alt="heure"/>❻
 </body>
</html>
```

Figure 12-25
Déformation d'une image



Les transitions des transformations en CSS 3

Les transformations que nous venons de réaliser sont opérées de manière immédiate dans les navigateurs. Dans le domaine des effets graphiques, les nouveautés CSS 3 permettent d'effectuer une transition dans les transformations, ce qui donne un effet visuel plus agréable. Nous avons à notre disposition quatre nouvelles propriétés qu'il faut encore faire précéder des préfixes, sachant que pour l'instant seuls Firefox, Chrome, Safari et Opera les gèrent. Elles vont nous servir à définir les différents paramètres de la transition. La première propriété :

```
transition-property:prop1, prop2,…,propN | all;
```

établit la liste des propriétés CSS pour lesquelles on veut opérer une transition. La valeur `all` signifie que l'on peut utiliser toutes les propriétés susceptibles d'être l'objet d'une transition. Le tableau 12-2 en donne la liste.

La propriété suivante définit la durée de la transition en secondes :

```
transition-duration: Ns
```

La troisième propriété définit la vitesse de la transition :

```
transition-timing-function:linear | ease | ease-in | ease-out |ease-in-out;
```

Ses valeurs ont le rôle suivant :

- `linear` : la vitesse de la transition est constante du début à la fin ;
- `ease-in` : la vitesse de la transition augmente ;
- `ease-out` : la vitesse de la transition diminue ;
- `ease-in-out` : la vitesse de la transition est lente au début et à la fin.

Pour être franc, les tests effectués ne me permettent pas d'apprécier les différences entre les trois dernières valeurs et la valeur `linear` que nous utiliserons ici. Enfin, une dernière propriété :

```
transition-delay: Ns;
```

définit le temps en secondes avant que la transition commence. C'est aussi le temps qu'elle met pour disparaître si elle est réversible (déclenchée avec `:hover` par exemple). S'il nous fallait écrire ces propriétés avec les préfixes pour cinq navigateurs, cela nous ferait vingt lignes de code. Pour faciliter le codage, il existe, comme nous l'avons déjà vu par ailleurs, un raccourci qui les définit toutes en une ligne ; c'est la propriété `transition` dont la syntaxe est :

```
transition: prop || durée || fonction || délai ;
```

qui définit dans l'ordre `transition-property`, `transition-duration`, `transition-timing-function` et `transition-delay`, ce qui condense le code nécessaire.

Tableau 12-2. Propriétés susceptibles de faire l'objet d'une transition

background-color, background-image, background-position, border-bottom-color, border-bottom-width, border-color, border-left-color, border-left-width, border-right-color, border-right-width, border-spacing, border-top-color, border-top-width, border-width, bottom, color, font-size, font-weight, height, left, letter-spacing, line-height, margin-bottom, margin-left, margin-right, margin-top, max-height, max-width, min-height, min-width, opacity, outline-color, outline-offset, outline-width, padding-bottom, padding-left, padding-right, padding-top, right, text-indent, text-shadow, top, vertical-align, visibility, width, word-spacing, z-index.

Dans l'exemple 12-18, nous procédons en deux temps pour créer la transition pour une image (repère ❽). Tout d'abord nous définissons les caractéristiques de la transition, à l'aide des propriétés individuelles pour Firefox (repère ❶) et des raccourcis pour Opera, Chrome et Safari, Explorer et la version standard (repères ❷ à ❺). Ensuite nous définissons ce que doit être la transformation pour l'image quand elle sera survolée par le curseur, à savoir un changement de dimensions (repère ❻) et une rotation de 360 degrés (repère ❼). Le résultat sera visible sur le site du livre.

Exemple 12-18 Transition pour un agrandissement

```
<!DOCTYPE html>
<html>
 <head>
  <meta http-equiv="Content-type" content="text/html;charset=UTF-8" />
  <title>Effets de transition</title>
  <style type="text/css">
   img{
        width:60px; height:40px;
```

```
        /* Propriétés individuelles */❶
        -moz-transition-property: all;
        -moz-transition-duration: 3s;
        -moz-transition-timing-function:linear;
        -moz-transition-delay:1s;
       /* Propriétés raccourcies */
       -o-transition: all 3s linear 1s;❷
       -webkit-transition: all 3s linear 1s;❸
       -ms-transition: all 3s linear 1s;❹
       transition: all 3s linear 1s;❺
       }
    img:hover {
       /* Agrandissement */
       width: 800px;height:500px;❻
       /* Rotaion de 360 degrés */❼
       -moz-transform:rotate(360deg);
       -o-transform:rotate(360deg);
       -webkit-transform:rotate(360deg);
       -ms-transform:rotate(360deg);
       transform:rotate(360deg);
                 }
   </style>
  </head>
  <body>
   <img  src="Romy2011.jpg" alt="Romy"/>❽
  </body>
 </html>
```

Exercices

Exercice 1

Incorporer trois images dans une page en définissant pour l'élément `<img />` les propriétés CSS `width` et/ou `height` (sans utiliser les attributs de même nom). Que se passe-t-il si les dimensions intrinsèques des images dépassent ces valeurs ?

Exercice 2

Dans l'exercice précédent, comment procéder au dimensionnement CSS de façon que chaque image occupe la place qui lui est nécessaire d'après les dimensions du fichier image ?

Exercice 3

Si les dimensions d'une image sont définies en pourcentage de celles de son conteneur, et que ses proportions largeur/hauteur sont inconnues, comment procéder pour qu'elle ne soit pas déformée ?

Exercice 4

Inclure trois éléments `<div>` contenant du texte, l'un dans l'autre, et définir la largeur à 70 % du précédent pour chacun d'eux. Le premier doit avoir 800 pixels de haut et les suivants doivent correspondre à 80 % de la hauteur du précédent.

Exercice 5

Créer deux paragraphes d'une hauteur de 300 pixels et d'une largeur de 700 pixels. Y inclure un texte très long et gérer son débordement afin qu'il soit entièrement lisible.

Exercice 6

Créer cinq titres de niveau 2 et les afficher sous forme de liste (voir la propriété `display`).

Exercice 7

Dans un élément `<div>`, inclure un élément `<span>` contenant du texte et lui donner le style bloc.

Exercice 8

Créer un menu vertical dont les éléments sont des liens `<a>`.

Exercice 9

Créer une page contenant un paragraphe incluant du texte et deux éléments `<img />` qui se suivent. Faire flotter les images, la première à gauche et la seconde à droite.

Exercice 10

Reprendre l'exercice précédent et empêcher le flottement de la deuxième image.

Exercice 11

Placer trois images de tailles initiales différentes dans une page. Écrire les styles pour qu'elles s'affichent avec la même taille. Ensuite, les positionner afin d'obtenir un effet de cascade avec un décalage horizontal et vertical constant pour chaque image par rapport à la précédente. L'utilisateur doit pouvoir mettre chacune d'elles au premier plan en cliquant dessus (voir la propriété `z-index`).

Exercice 12

Créer une mise en page à trois colonnes de largeurs respectives 20 %, 65 % et 15 %. La première et la troisième doivent contenir respectivement un menu et une liste de liens créés à partir d'images. La colonne centrale doit posséder un contenu éditorial.

Exercice 13

Créer une mise en page selon le modèle de la figure ci-dessous :

La colonne de gauche (repère ❶) a une largeur de 200 pixels et le bandeau (repère ❷) une hauteur de 150 pixels. Le bandeau contient le titre du site, la colonne de gauche un menu et la zone principale (repère ❸) du texte et des images au choix. Le premier lien du menu doit afficher une page ayant la même structure et le même contenu dans les zones ❶ et ❷, mais un contenu éditorial différent dans la zone ❸.

Exercice 14

Reprendre l'exemple précédent de façon que les zones ❶ et ❷ soient fixes dans la fenêtre du navigateur.

Exercice 15

Insérer un texte long dans une page et réaliser un affichage sur quatre colonnes.

Exercice 16

Insérer trois images en réglant leur taille à une valeur beaucoup plus petite que leur taille normale, et permettre leur redimensionnement par le visiteur.

Exercice 17

Effectuer la rotation d'un texte de 90 degrés.

Exercice 18

Effectuer la rotation de 90 degrés d'une image avec une transition de 4 secondes.

Exercice 6

À quoi sert l'élément `<head>` ? Quels sont ses éléments enfants ? Peuvent-ils être employés plusieurs fois dans le même document ?

Exercice 7

Quel élément est obligatoire dans l'élément `<head>` ? À quoi sert-il ?

Exercice 8

Écrire le code nécessaire à la liaison d'une feuille de style avec un document.

Exercice 9

Écrivez le code nécessaire à la liaison d'un document externe contenant des scripts JavaScript avec un document.

Exercice 10

Comment déclarer les mots-clés associés au site ? Quelle est l'utilité de cette déclaration ? Écrivez-en un exemple.

Exercice 11

Quel est le rôle de l'élément `<base />` ? Écrivez-en un exemple.

Exercice 12

Peut-on écrire le code suivant dans une page ?

```
<body>
 Bienvenue dans notre site
 <h1>Le site du HTML 5 et de CSS 3</h1>
</body>
```

Exercice 13

Peut-on inclure les éléments `<tbody>`, `<form>` et `<li>` directement dans l'élément `<body>` ?

Exercice 14

Écrivez un script qui affiche un message d'alerte quand l'utilisateur arrive sur le site.

Exercice 15

Écrivez le code CSS suivant à l'endroit adéquat :

```
body {background-color:white;color:green;font-size :20px}
```

Incluez ensuite un texte dans la page et testez le résultat. Vous devez obtenir un fond rouge, un texte bleu avec des caractères de 20 pixels.