6

Résolution d'erreurs

Ce chapitre s'aventure dans le monde obscur et redoutable des différences d'affichage selon les navigateurs du marché. Il parcourt les contrées malfamées des *hacks* et autres « bidouilles », du mécanisme étrange de *HasLayout* et de toutes les solutions permettant d'harmoniser des pages sur l'ensemble des navigateurs.

Connaître le rendu par défaut des éléments

Il ne s'agit pas d'un scoop : un site bien conçu, voire validé par l'outil du W3C, ne sera pas obligatoirement affiché de la même façon partout. En effet, chaque navigateur a ses spécificités (marges différentes sur les éléments) et ses erreurs de rendu.

Outils de vérification

Sur www.alsacreations.com, nous proposons un outil conçu par Julien Royer et spécialisé dans cette tâche : en accédant à la page alsacreations.com/static/rendu/, un tableau contenant la plupart des balises HTML usuelles est créé selon le navigateur que vous employez. Pour chacun de ces éléments, les valeurs par défaut des propriétés `display`, `margin`, `padding`, `font-size` et `font-weight` sont renseignées. Si vous visitez cette page à l'aide d'un autre navigateur, soyez sûr que certaines valeurs seront différentes !

D'une manière plus générale, le site web IEcss.com permet d'afficher les styles par défaut de tous les éléments HTML sur l'ensemble des versions d'Internet Explorer depuis IE6 jusqu'à IE9 et de comparer leurs différences. Il s'agit d'un site plus qu'intéressant pour comprendre les disparités d'affichage d'un navigateur à un autre (figure 6-1).

Figure 6-1

IEcss.com

Internet Explorer User Agent Style Sheets

The UA Style Sheet is a simple set of css styles that each web browser uses before any other css styles are applied.

This chart lists and compares the different default style sheets used to render HTML in the four major versions of Internet Explorer; IE6, IE7, IE8, and IE9 Platform Preview.

You can download each of these UA stylesheets by using the links at the top of this chart.

	IE6	IE7	IE8	IE9
a	color: #00F; text-decoration: underline;	color: #06C; text-decoration: underline;		
a:visited	color: #800080;			
address	display: block; font-style: italic;			
b	font-weight: bold;			
bdo	direction: rtl; unicode-bidi: bidi-override;			
blockquote	display: block; margin: 14pt 30pt;		display: block; margin: 1em 40px;	
body	display: block; margin: 15px 10px; zoom: 1;		display: block; margin: 8px; zoom: 1;	

Et si ce n'était pas une erreur ?

Ce n'est pas forcément une bonne nouvelle pour vous, mais au fur et à mesure de leurs versions, les navigateurs sont de plus en plus stables, de plus en plus conformes aux spécifications et de moins en moins bogués. Le corollaire est que, par déduction, la source du problème est souvent… vous.

En effet, les erreurs que l'on nous soumet sur le forum Alsacréations ne sont généralement pas dues à des manquements des navigateurs, mais plus souvent à des lacunes humaines :

- une mauvaise compréhension des schémas de positionnement (à savoir qu'il est attendu qu'un élément hors flux ou flottant « dépasse » de son parent) ;
- des imbrications d'éléments non autorisées (par exemple, un élément HTML de type `block` au sein d'un élément `inline` ou d'un paragraphe) ;
- des erreurs de codage (balise fermante oubliée, séparateur point-virgule omis en CSS, etc.) ;
- une mauvaise compréhension du modèle de boîte, par exemple :
  - les marges internes et les bordures s'ajoutent à `width` lorsqu'il s'agit de calculer l'espace occupé par l'élément ;
  - tous les éléments HTML de type bloc (à l'exception de `<div>`) disposent de marges externes non nulles par défaut ;
  - les fusions de marges modifient l'espacement entre deux éléments.

En résumé, il est parfois plus sage de commencer par bien consolider ses acquis plutôt que de blâmer les agents utilisateurs dont le seul tort est de respecter scrupuleusement les spécifications officielles.

COMPRENDRE Espaces sous les images

Certains comportements d'affichage, aussi curieux qu'ils paraissent, sont pourtant parfaitement conformes. C'est le cas du célèbre espace de quelques pixels que l'on peut observer sous les images (`<img>`), impossible à évacuer en redéfinissant les `margin`, `padding` et autre `line-height` à zéro.

L'explication est pourtant simple : `<img>` est un élément en ligne de type `inline-block` qui se positionne par défaut sur la ligne de texte (*baseline*), donc laisse un espace en dessous pour les lettres telles que « p » ou « q ».

Pour remédier à ce comportement, il est possible d'agir sur la propriété `vertical-align` en lui conférant la valeur `bottom`, ou encore de modifier le mode de rendu de l'image à l'aide de `display: block`.

Faut-il utiliser les hacks ?

Un *hack* (ou « bidouille », en français) est une technique fondée sur l'empirisme, destinée à éviter, au cas par cas, certaines différences de rendu entre les navigateurs et permettre aux plus défectueux d'aboutir à nos fins, soit en détournant une propriété de son usage, soit en utilisant des codes supplémentaires pour pallier les manques.

Les astuces de ce genre sont généralement déconseillées, car elles reposent sur une déficience temporaire d'une version de navigateur qui peut être corrigée du jour au lendemain.

Exemples de hacks

L'un des hacks les plus connus est le *simplified box model hack*, dont l'astuce consiste à insérer une barre oblique inverse (\) au sein du nom de la propriété, par exemple : `w\idth: 100px`. Cette manœuvre avait au départ la bonne idée de n'être prise en compte que par Internet Explorer 6 et 7, ce qui pouvait arranger les concepteurs de pages en mode Quirks :

```
div {
  width: 80px;
  padding: 10px;
  w\idth: 100px; /* ignoré sauf sur IE6 et IE7 */
}
```

Certains hacks affichent des syntaxes tellement élaborées que l'on peut se poser des questions sur l'état psychologique de leur inventeur, telle la déclaration `/*/*//*/ color:green; /* */`, qui n'est reconnue que par Netscape 4 !

ALLER PLUS LOIN Le musée des hacks

Le site web Centricle recense tous les hacks trouvés par les concepteurs web ainsi que leur reconnaissance par les différentes versions de navigateurs :

- http://centricle.com/ref/css/filters/

Risques pour l'avenir

Un hack est non seulement un disgracieux détournement de syntaxes conformes, mais aussi et surtout, une source de conflit pour le futur. Nul ne peut deviner aujourd'hui l'avenir d'une syntaxe « mutante » : elle peut être corrigée par le navigateur déficient ou, pire, être intégrée aux spécifications et devenir parfaitement conforme et reconnue par tous.

Par exemple, le *simplified box model hack* évoqué précédemment est aujourd'hui pris en considération par un large ensemble de navigateurs contemporains tels que Firefox, Chrome, Opera et IE8. Cela signifie que les syntaxes telles que `w\idth: 100px` ne sont plus du tout ignorées par ces navigateurs et que tous les sites web ayant eu recours à cette ruse en espérant ne cibler que les anciennes versions d'Internet Explorer en font dorénavant les frais !

Hacks à méditer ?

Au sein de la jungle des hacks existants ou ayant existé, tous ne sont pas systématiquement bons à jeter. Il se peut que dans certains cas, très spécifiques et parfaitement délimités, l'usage de ces stratagèmes soit parfaitement légitime. Quoi qu'il en soit, réservez-les pour les situations extrêmes, quand toutes les autres possibilités ont dû être écartées et n'employez que des hacks considérés comme bénins.

Parmi ces entourloupes jugées moins risquées, certaines s'appliquent à des propriétés et d'autres à des sélecteurs. Selon vos besoins, l'une ou l'autre pourrait s'avérer nécessaire.

Hacks de propriétés

Les deux syntaxes suivantes sont relativement stables, car elles ne sont actuellement prises en compte par aucun navigateur à l'exception de ceux mentionnés. Cependant, la probabilité pour que ces grammaires fassent un jour partie des spécifications CSS n'est pas nulle. À vous de prendre vos responsabilités si vous les employez :

- `_propriété` (exemple :`_display: inline`) – il s'agit du *Underscore hack*, aujourd'hui uniquement reconnu par Internet Explorer 6 (et précédents) ;
- `*propriété` (exemple : `*display: inline`) – nommé *Asterisk hack*, il n'est reconnu que par IE6 et IE7.

Une autre syntaxe, plus connue, est parfaitement exploitable si vous aviez à cibler Internet Explorer uniquement : le mot-clé `!important`. En effet, les versions 5 et 6 de ce navigateur interprètent assez mal cette fonctionnalité : lorsque deux propriétés se contredisent au sein d'un même bloc de déclaration, IE6 ne reconnaît pas celle munie du mot-clé `!important`.

```
div {
  width: 100px!important; /* pour tous les navigateurs */
  width: 80px; /* pour IE5 et IE6 */
}
```

Hacks de sélecteurs

En ce qui concerne les hacks de sélecteurs, un seul est digne d'intérêt : le *star HTML bug*, qui s'écrit sous la forme `* html sélecteur {…}`, uniquement reconnu par IE6, et qui se base sur un défaut de ce navigateur : il compte un élément invisible parent de `<html>` dans son modèle de rendu.

Cibler les navigateurs récents à l'aide de sélecteurs avancés

Si le terme de *hack* et son usage vous mettent mal à l'aise – et je vous comprends – vous serez sans doute plus séduit par une autre pratique bien plus consciencieuse : utiliser volontairement des sélecteurs dits « avancés » pour cibler uniquement les navigateurs récents.

Cibler à partir d'Internet Explorer 7

Pour illustrer ce concept, prenons une syntaxe telle que celle-ci :

```
html > body div {…}
```

Comme nous l'avons vu en début d'ouvrage, le symbole `>` désigne le sélecteur d'enfant. La règle s'applique par conséquent « aux éléments `<div>` descendants de `<body>`, lui-même enfant de `<html>` ». Cette lourdeur peut paraître inutile de prime abord, puisque `<html>` et `<body>` sont forcément ancêtres de tous les éléments du document.

L'astuce et l'intérêt de cette syntaxe demeurent dans le fait que ce sélecteur, parfaitement valide, n'est reconnu et appliqué qu'à partir d'Internet Explorer 7. Cela permet, dans un souci d'amélioration progressive, d'attribuer des styles qui ne seront pas pris en compte par IE6 et ses ancêtres.

Cibler à partir d'Internet Explorer 8

Le sélecteur CSS 3 `:lang()` a l'avantage d'être reconnu non seulement par l'ensemble des agents utilisateurs récents, mais également à partir d'Internet Explorer 8.

Il rend donc possible la détection de tous les navigateurs actuels en basant sa syntaxe sur un sélecteur qui débuterait par `:lang(fr)`. Cette astuce suppose bien entendu que l'attribut HTML `lang` soit appliqué à un élément de structure principal (généralement la balise `<html>`) et qu'il ait pour valeur `fr` !

Dans notre quête du positionnement idéal, cette méthode offre, par exemple, la possibilité d'employer le modèle de rendu CSS en tableau, tout en proposant une alternative aux anciens navigateurs :

```
#toto { /* Pour tout le monde, dont IE6 / IE7 */
  float: left;
  width: 300px;
}
:lang(fr) #toto {  /* Pour les navigateurs modernes et IE8 */
  display: table-cell;
  float: none;
  width: auto;
}
```

Une autre éventualité : positionner à l'aide de `display: inline-block` tout en proposant une équivalence pour IE6 et IE7 :

```
#toto { /* Pour IE6/IE7 et les anciens navigateurs */
  display: inline;
  zoom: 1;
}
:lang(fr) #toto {  /* Pour les navigateurs modernes et IE8 */
  display: inline-block;
}
```

Bien que séduisantes, ces techniques de sélecteurs avancés sont à méditer au cas par cas, car il pourrait être plus judicieux d'utiliser un commentaire conditionnel ciblant IE7 ou IE8 et inférieur (voir partie suivante).

Préférer les commentaires conditionnels

Tous les navigateurs, selon leur ancienneté ou leur degré de conformité aux standards, ne prennent pas en charge les propriétés CSS de la même manière. Certains présentent des erreurs d'interprétation, quelques-uns reconnaissent les dernières nouveautés en CSS 3, d'autres non.

Nous avons constaté que, au fur et à mesure de l'évolution du navigateur, celui-ci devient de plus en plus conforme aux standards et les anciens hacks utilisés deviennent obsolètes. Ainsi, depuis l'arrivée d'IE7, nombre de contournements appliqués pour IE6 sont devenus inopérants car corrigés. Les concepteurs web ont dû reprendre une par une toutes leurs ruses pour convenir à la nouvelle version.

Pour remédier à ce problème, il existe une solution plus robuste et plus pérenne : les commentaires conditionnels. Il s'agit d'un mécanisme propre à Internet Explorer Windows, né avec la version IE5, et qui permet d'inclure dans une page HTML, de manière valide, une portion de code qui ne sera lue et interprétée que par IE, ou par l'une ou l'autre de ses versions.

Microsoft lui-même conseille aux concepteurs web d'employer la méthode des commentaires conditionnels afin d'éviter les hacks : « Nous vous recommandons de bien vouloir modifier vos pages afin de ne plus utiliser les 'hacks' CSS. Ceux qui souhaitent cibler ou éviter Internet Explorer pourront dorénavant employer les commentaires conditionnels. » *(Marjus Mielke, Microsoft, octobre 2005.)*

Fonctionnement

Les commentaires conditionnels se présentent comme des instructions dotées d'une condition (`if`) et qui peuvent se placer à n'importe quel endroit du document (X)HTML.

La syntaxe la plus simple est la suivante.

Partie HTML

```
<!--[if IE]>
  ici votre code HTML réservé à IE
<![endif]-->
```

Usage pratique

L'un des avantages majeurs de ce mécanisme est de pouvoir cibler les versions d'Internet Explorer qui devront suivre les instructions contenues au sein des commentaires conditionnels.

Partie HTML

```
<!--[if IE 6]> pour IE6 uniquement <![endif]-->
<!--[if gt IE 6]> pour les versions supérieures à IE 6 <![endif]-->
<!--[if gte IE 6]> pour les versions supérieures ou égales à IE 6 <![endif]- >
<!--[if lt IE 8]> pour les versions inférieures à IE 8 <![endif]-->
<!--[if lte IE 8]> pour les versions inférieures ou égales à IE 8 <![endif]-->
```

Il est par ailleurs possible d'incorporer des opérateurs logiques tels que `&` (et) ou encore `|` (ou).

Partie HTML

```
<!--[if (gte IE 6)&(lte IE 8)]> entre les versions IE 6 et IE 8 uniquement <![endif]
-->
<!--[if (IE 6)|(IE 8)]> pour les versions IE 6 ou IE 8 uniquement <![endif]-->
```

Il est même envisageable de cibler les navigateurs qui ne sont *pas* Internet Explorer :

Partie HTML

```
<!--[if !IE]><!--> pour les navigateurs non IE <!--<![endif]-->
```

Notez que la syntaxe est assez alambiquée dans ce dernier cas de figure, pour des raisons d'échappement de caractères et de validation du code.

Voici par exemple comment afficher un message d'information aux utilisateurs d'anciennes versions de navigateur (figure 6-2).

Figure 6-2

Message personnalisé selon le navigateur

Partie HTML à placer en début de <body>

```
<!--[if lte IE 6]>
  <div id="alert-ie6"><strong>Attention ! </strong> Votre navigateur (Internet
  Explorer 6) présente de sérieuses lacunes en terme de sécurité et de performances,
  dues à son obsolescence (il date de 2001).<br />En conséquence, ce site sera
  consultable mais de manière moins optimale qu'avec un navigateur récent.</div>
<![endif]-->
```

Dans notre quête de compatibilité maximale sur le navigateur de Microsoft, nous allons principalement nous servir de ce mécanisme pour faire un lien vers une feuille de styles corrective dédiée. Cette seconde feuille CSS, chargée à la suite du fichier de styles principal, aura pour but d'écraser et de rectifier au cas par cas les règles générales mal reconnues par IE.

Partie HTML

```
<link rel="stylesheet" href="styles.css" type="text/css" />
<!--[if lt IE 8]>
<link rel="stylesheet" href="styles_ie.css" type="text/css" />
<![endif]-->
```

Cet usage des commentaires conditionnels est devenu très vite un standard de fait, pratiqué par une majorité de la communauté des concepteurs web. En effet, il présente un certain nombre d'avantages, dont celui de la pérennité et de la propreté, le code demeurant valide contrairement aux hacks.

Classe conditionnelle pour Internet Explorer

En 2008, le développeur américain Paul Irish s'est penché sur le mécanisme des commentaires conditionnels et en a listé quelques désagréments :

- Les commentaires conditionnels requièrent une ou plusieurs requêtes HTML, ce qui est néfaste en terme de performances.
- S'ils sont appelés dans l'élément `<head>`, le temps d'affichage de la page est soumis au chargement complet des styles afférents.
- Le fait de multiplier les feuilles de styles et les sélecteurs identiques dans plusieurs fichiers ne facilite guère la relecture et la correction des erreurs.

L'argument des baisses de performances est non négligeable sur les versions IE6 et IE7, car il s'agit justement de navigateurs vieillots nécessitant des optimisations constantes et nombreuses en vue de ne pas handicaper leurs temps de calcul et d'affichage.

La solution proposée par Paul Irish a été de créer un nom de classe spécifique à l'élément `<body>`, via un commentaire conditionnel et sans nécessiter d'appel vers une feuille de styles.

Voici la version courte de sa méthode de *classe conditionnelle* :

Partie HTML

```
<!--[if lte IE 7]> <body class="ie7"> <![endif]-->
<!--[if !IE]><!--> <body> <!--<![endif]-->
```

Le principe est simple : sur les versions inférieures ou égales à Internet Explorer 7, le corps de page du document s'écrira `<body class="ie7">` ; sur tous les autres navigateurs, il s'agira simplement de `<body>`. Il devient alors aisé d'appliquer une règle spécifique à IE6 et IE7 :

Partie CSS

```
#kiwi { /* Pour tout le monde */
  display: inline-block;
}
.ie7 #kiwi { /* Pour IE6 et IE7 */
  display: inline;
  zoom: 1;
}
```

Cette méthode – qui, contrairement à ce que l'on pourrait croire, est parfaitement valide – commence à faire son chemin parmi les développeurs web aguerris. Pour les autres, la syntaxe demeure un tantinet rebutante.

HasLayout chez Internet Explorer

Un mécanisme propriétaire

Le *HasLayout* est un ancien mécanisme complexe et interne d'Internet Explorer, apparu sur les versions IE5, IE6 et IE7 (ainsi que sur IE8 en « mode de compatibilité »). Il disparaît sur IE8 standard et IE9.

Pour faciliter et accélérer l'affichage des pages web, Microsoft a jugé utile de distinguer en amont certains éléments HTML destinés à être dimensionnés et positionnés. Ces éléments triés sur le volet sont automatiquement dotés d'un attribut JavaScript en lecture seule, indiquant s'ils bénéficient ou non de cette faculté de *Layout* (structure de page) : c'est le concept du *HasLayout* (littéralement, « qui possède le *Layout* »).

S'il améliore les performances de rendu sur Internet Explorer, cet obscur mécanisme est également – et malheureusement – lié à de nombreuses erreurs d'affichage sur ce navigateur et ses conséquences se font sentir dans nos mises en page web.

Avoir le Layout

Ce concept est inhérent au moteur de rendu d'Internet Explorer et n'existe pas sous forme de propriété ou règle CSS. Selon le navigateur de Microsoft, un élément possède le Layout (*HasLayout*) ou ne le possède pas. JavaScript permet de lire cette valeur booléenne, telle un interrupteur positionné sur *true* (vrai) ou *false* (faux).

Certains éléments sont dotés de Layout par défaut. Il s'agit de `<body>`, `<table>`, `<td>`, `<img>`, `<hr>`, `<input>`, `<select>`, `<textarea>`, `<button>`, `<object>` ou encore `<applet>`. Aucun des autres éléments HTML – et ils sont nombreux – ne dispose de cet interrupteur de Layout sur la valeur *true*.

Donner et ôter le Layout

Bien qu'il s'agisse d'une valeur JavaScript en lecture seule, Microsoft explique sur son site technique (msdn.microsoft.com), qu'il est possible de conférer le Layout aux éléments qui ne le possèdent pas au départ, dans le but de corriger de nombreuses incohérences sur Internet Explorer.

CSS et la propriété zoom

Contre toute attente, la seule méthode permettant d'attribuer le Layout à un élément HTML ne relève pas du langage JavaScript, mais de… CSS.

En appliquant n'importe laquelle des déclarations CSS suivantes sur un élément dénué de Layout par défaut, il acquiert ce fameux privilège :

- `heigth` ou `width` : avec une valeur autre que `auto` ;
- `min-width` ou `min-height` (IE7 minimum) : n'importe quelle valeur (même 0) ;
- `max-width` ou `max-height` (IE7 minimum) : n'importe quelle valeur ;
- `float: left` ou `float: right` ;
- `position: absolute` ;
- `display: inline-block` ;
- `overflow: hidden, scroll` ou `auto` (depuis IE7) ;
- `zoom: valeur` (propriétaire Microsoft).

La liste des propriétés réellement applicables sans dénaturer l'objet est bien plus maigre qu'on ne le croit : il est généralement hors de question de sortir l'élément du flux (`position`, `float`) ou de modifier ses dimensions (`width`, `height`, `max-width`…). La propriété `overflow` peut causer de gros soucis d'affichage en cas de débordements et `display: inline-block` modifie le rendu et n'est reconnue par IE que sur un élément de type `inline` au départ.

Toutes ces considérations prises en compte, il ne reste finalement plus qu'une seule syntaxe méconnue – et pour cause, elle ne figure pas dans les spécifications – la propriété `zoom`.

Inventée par Microsoft, la propriété `zoom` est l'ancêtre de la déclaration CSS 3 `transform: scale` produisant un effet de grossissement sur l'élément. Agissant sous forme de ratio, l'expression `zoom: 2` se lit comme un rapport de 2/1, soit une échelle correspondant au double de la taille initiale.

Pour conférer le Layout sans modifier la structure, la taille ou les dimensions de l'élément, la solution la plus salutaire sera de lui attribuer la déclaration `zoom: 1`.

Si vous tenez à conserver un fichier CSS validé par l'outil automatique du W3C, vous placerez cette syntaxe propriétaire au sein d'une feuille CSS séparée et appelée via un commentaire conditionnel.

> **ATTENTION Restrictions pour les éléments de type inline**
>
> Pour compliquer encore un peu la situation, sachez que les propriétés `width` et `height` ne confèrent pas le Layout aux éléments de type HTML `inline` ou dotés d'une déclaration `display: inline`.

Supprimer le Layout ?

On ne peut pas supprimer le Layout des éléments qui en sont dotés par défaut, cependant, rien ne nous empêche de l'annuler pour ceux qui l'ont hérité via CSS, en redéfinissant les valeurs par défaut des propriétés, par exemple `width` et `height` à la valeur `auto`, `zoom` à `normal`, `position` à `static` ou encore `float` à `none`.

Du Layout et des erreurs

Le mécanisme de *HasLayout* est complexe à assimiler, pas uniquement parce qu'il s'agit d'une « cuisine interne » à Microsoft, mais aussi et surtout en raison de ses effets pervers : dans certains cas de figure, avoir le Layout corrige instantanément un défaut d'affichage sur Internet Explorer ; dans d'autres, cela crée des problèmes de rendu. Tout dépend du contexte. Pire, le comportement peut survenir d'une page à l'autre d'un site, sans raison apparente.

Erreurs corrigées en donnant le Layout

Le nombre d'erreurs ou d'écarts vis-à-vis des spécifications CSS recensés pour Internet Explorer 6 est assez impressionnant (par chance, l'équipe de Microsoft en a résolu énormément dans la version 7). En conséquence, il est parfois difficile de détecter si une erreur est due à la présence ou à l'absence de Layout sur un élément.

Retenez toutefois un chiffre éloquent : au moins 50 % des dysfonctionnements visuels sur IE6 et IE7 sont à mettre au crédit du mécanisme *HasLayout*. Il peut s'agir de problèmes de rendu très variés et souvent surprenants : des décalages de flottants, des espaces au sein des listes, des disparitions d'arrière-plan voire d'éléments entiers, ainsi que des problèmes de positionnement en général.

Voici une illustration représentative de ce phénomène déroutant. L'exemple suivant décrit une liste de liens ayant bénéficié de la règle `display: block` (figure 6-3) :

Partie HTML

```
<ul>
  <li><a href="#">Accueil</a></li>
  <li><a href="#">Société</a></li>
  <li><a href="#">Contact</a></li>
</ul>
```

Partie CSS

```
ul {background: #CCCC66; }
li {
  list-style: none;
  background: #6B9A49;
}
li a {
  display: block;
  color: white;
}
```

Affichée sur Internet Explorer 6, cette page présente une particularité assez symptomatique : un espace libre correspondant à la hauteur d'un caractère apparaît sous chacun des liens !

Figure 6-3

Erreur dans le rendu d'une liste de liens sous IE6

Une erreur d'affichage de ce type pour le moins inattendu (et heureusement corrigé sur IE7) laisse penser qu'il peut s'agir d'un problème de *HasLayout*.

En effet, en conférant le Layout à l'élément `<a>`, le problème de rendu disparaît instantanément (figure 6-4) :

```
li a {
  display: block;
  zoom: 1;
}
```

Figure 6-4

Erreur corrigée en attribuant le Layout

Erreurs dues au Layout

Les exemples d'altérations visuelles dues à l'absence de Layout sont nombreux et nous serions presque tentés d'appliquer la solution radicale qui est de conférer cet attribut à l'ensemble des éléments de la page, ainsi :

```
* {zoom: 1;}
```

Malheureusement, bénéficier du Layout est loin d'être la solution à toutes les incohérences d'affichage sur Internet Explorer. C'est même souvent le contraire : *HasLayout* confère une sorte de « toute-puissance » aux éléments, qui n'en font qu'à leur tête.

Parmi les cas d'erreurs dues à la présence du Layout, citons :

- Les éléments dimensionnés s'élargissent à tort, pour s'adapter à la largeur ou la hauteur de leurs contenus.
- La fusion de marges n'opère plus sur les éléments dotés de Layout.
- Les éléments hors flux occupent toute la largeur du référent alors qu'il devraient se restreindre à la largeur de leur contenu.
- Les éléments flottants ne dépassent plus de leur parent si ce dernier possède le Layout.

Voici une situation où la présence de Layout engendre un rendu non conforme aux spécifications. Dans l'exemple qui suit, notre structure comporte un paragraphe `<p>` au sein d'un bloc `<div>` positionné en absolu :

Partie HTML

```
<div>
  <p>Un kiwi</p>
</div>
```

Partie CSS

```
div, p {margin: 0;}
div {
  position: absolute;
  background: green;
}
p {color: white;}
```

Tel que les spécifications le préconisent, le `<div>` retiré du flux n'occupe pour largeur que la taille de son contenu, c'est-à-dire le texte « Un kiwi » (figure 6-5).

Figure 6-5

Pas d'erreur sans Layout

Toutefois, le paragraphe enfant de `<div>` qui se verrait octroyer le Layout, par exemple avec la règle `p {zoom: 1;}`, hérite, sur IE6 exclusivement, d'un pouvoir tel qu'il pousse son parent de manière à ce que l'ensemble occupe toute la largeur (figure 6-6).

Figure 6-6

Erreur due au Layout

L'inventaire des problèmes liés à la présence ou l'absence de Layout est si vaste qu'il est généralement nécessaire de traiter chaque situation au cas par cas. Sachez toutefois que si vous êtes confrontés à un dysfonctionnement caractéristique des versions 6 et 7 d'Internet Explorer, il y a de fortes probabilités qu'il s'agisse d'un problème de *HasLayout*. Et vous avez à présent les cartes en main pour le contrer.

Petite méthodologie de résolution d'erreurs

En théorie, résoudre un problème d'affichage se passe de la manière suivante : on commence par cibler et isoler l'élément problématique, on établit un diagnostic, puis on corrige les défauts. En pratique, c'est souvent un peu plus acrobatique et périlleux, d'autant plus qu'une erreur se présente rarement seule. Voyons comment procéder de la façon la plus méthodique possible.

Isoler l'élément

Un problème d'affichage est dû en général à des incompatibilités de navigateurs ou à des marges par défaut différentes selon les navigateurs. La première étape à suivre consiste donc à isoler l'élément qui ne se comporte pas comme vous l'aviez prévu.

Appliquer une couleur de fond

Actuellement, les sites web sont presque tous structurés à l'aide des balises `<div>`, `<table>`, `<hn>`, `<p>` et `<ul>`.

Les CSS offrent un moyen très simple d'isoler ces différents éléments : en leur attribuant une couleur de fond, vous bénéficierez immédiatement d'un visuel global de la structure de votre mise en page, sans toucher au contenu ni au code HTML.

En commençant votre feuille de styles avec tout ou partie des déclarations ci-après, vous aurez un aperçu de l'espace exact occupé par chacun des conteneurs. Les tables et cellules seront en gris, les `<div>` en beige, les éléments de titre ou de paragraphes auront une couleur kaki, etc. Ceci permettra très rapidement de mettre le doigt sur des décalages ou des erreurs d'affichage concernant l'un ou l'autre de ces éléments.

Déclarations CSS pour appliquer une couleur de fond

```
table {background-color: lightgray}
td, th {background-color: gray}
div {background-color: beige}
form {background: bisque}
input, select, textarea {background-color: salmon}
h1, h2, h3, h4, h5, h6, p {background-color: darkkhaki}
ul {background-color: lightsteelblue}
li {background-color: silver}
etc.
```

Cette étape permet également de localiser les éventuelles occurrences de fusion de marges qui peuvent occasionner certains décalages verticaux entre les blocs possédant des marges externes.

MEA CULPA Et si on ajoutait aussi une bordure ?

Dans mon livre précédent, *CSS 2 : pratique du design web*, j'évoquais également l'alternative consistant à entourer les éléments sus-cités par une bordure colorée d'un pixel. J'évite dorénavant cette démarche pour une raison simple : une bordure, même de 1 px, va s'ajouter dans le calcul de la largeur et de la hauteur de l'élément, ce qui aura pour conséquence de créer des décalages d'affichage en plus des erreurs que nous avions à corriger !

Si cette technique de couleur d'arrière-plan ne porte pas ses fruits, tentez alors de masquer les éléments un par un (soit en CSS, à l'aide d'une règle telle que `élément1, élément2 {display: none;}`, soit en HTML, en coupant temporairement des sections du code).

Adopter Firebug

Certains outils sont indispensables dans la panoplie du parfait petit intégrateur web, notamment pour cibler et corriger les différences d'affichage entre les navigateurs. Au sein de la longue liste de logiciels ou d'extensions utiles, l'un d'entre eux est souvent évoqué comme étant irremplaçable au sein de notre communauté : Firebug.

Cette extension est disponible sur plusieurs navigateurs dont Firefox et Chrome. Elle permet en un clic d'avoir une vision exhaustive de l'ensemble des propriétés appliquées – ou non – à un élément, de visualiser sa boîte et les différentes composantes (`width`, `padding`, `margin`, `border`), mais aussi et surtout d'agir sur toutes les propriétés et valeurs : modifier une valeur ou une propriété, annuler une propriété, ajouter une déclaration, etc.

Avec un minimum de pratique, Firebug deviendra très vite votre outil préféré et vous fera économiser de longues heures de tests et de débogage.

> **OUTILS Et si vous n'utilisez pas Chrome ou Firefox ?**
>
> Firebug est un outil de débogage extraordinaire, mais il ne peut pas être installé partout. Sachez cependant que d'autres outils natifs existent sur tous les navigateurs : les inspecteurs d'éléments sur Internet Explorer 8, Chrome et Safari ou encore l'excellent Dragonfly sur Opera.

S'informer

Comme tout bon professionnel ou passionné que vous êtes, je vous conseille vivement de pratiquer une veille technologique constante sur les méthodes et découvertes dans le monde de la conception web.

Cet ouvrage ne peut que vous enseigner les bases de certaines techniques, ainsi que des pistes à suivre, mais sa portée demeure limitée. Vous pensez tout savoir sur les CSS ? Détrompez-vous. Vous croyez maîtriser le positionnement flottant ? Très bien. Et si je vous demandais ce que vous évoquent les célèbres erreurs de flottements recensées sur le site PositionIsEverything.net : *Float model*, *Double margin*, *Escaping floats*, *Peek-a-boo*, *Guillotine*, *Three pixel jog*, *Multiple opposing floats* (Opera) ou encore les quelques défauts sur IE5 Mac ?

Pas évident, n'est-ce pas ?

Voilà pourquoi il est important de toujours s'informer auprès de sites dédiés ou de forums de discussion spécialisés (tels que forum.alsacreations.com) et de suivre les informations quotidiennement via son agrégateur RSS ou un compte Twitter (voir la partie Ressources en annexe). Le Web est vaste, vous y trouverez sans aucun doute la source et le moyen de contrer votre erreur d'affichage.

Corriger l'erreur

À présent que l'élément est isolé, plusieurs outils faciliteront le diagnostic et le dépannage : la validation des fichiers, l'application d'un Reset CSS temporaire, la prise en compte des priorités des sélecteurs, ou encore l'attribution du Layout.

Valider ses fichiers

Nous accomplissons souvent la validation des pages par habitude, ou parce qu'elle est exigée dans le cahier des charges du client. Et pourtant c'est une première étape absolument nécessaire pour au moins deux raisons essentielles :

- Sans Doctype, si le Doctype est tronqué ou si des caractères apparaissent avant le Doctype, Internet Explorer passe en mode Quirks et dans son modèle de boîte erroné.
- Le Validateur HTML et le Validateur CSS indiquent que des éléments sont manquants, ou si le concepteur a oublié de fermer une balise. Internet Explorer est souvent plus restrictif que d'autres navigateurs sur ce point et affichera différemment vos pages.

Souvent, des erreurs proviennent de la « grammaire » et de la conception même du document. Soumettre sa page aux différents validateurs est une précaution initiale précieuse pour détecter ces problèmes. Passez vos documents aux validateurs automatiques (X)HTML et CSS sur le site du W3C pour effectuer votre premier diagnostic.

Reset CSS temporaire

Tous les éléments de type bloc (sauf `<div>`) possèdent des marges internes (`padding`) et externes (`margin`) par défaut. L'inconvénient est que cette valeur par défaut varie d'un navigateur à l'autre et engendre des rendus différents. Il s'agit certainement de l'explication la plus courante lorsque des décalages apparaissent entre les diverses plates-formes.

Le meilleur moyen d'identifier un problème de marges sur certains éléments est de... supprimer toutes les marges de tous les éléments. Le principe est d'utiliser le sélecteur universel (*), qui s'appliquera à toutes les balises, et de mettre les marges à zéro :

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

Commencez votre feuille de styles par cette déclaration. Si les décalages involontaires disparaissent, vous aurez détecté un problème de marges par défaut. Il vous appartient ensuite d'isoler l'élément qui provoque ce décalage, puis de supprimer cette règle de mise à zéro globale temporaire.

Attention à la priorité des sélecteurs

Dans un chapitre précédent traitant des bonnes pratiques CSS 2.1, j'avais mis l'accent sur la pondération accordée aux différents types de sélecteurs. Ainsi, les sélecteurs d'identifiants seront toujours prioritaires sur les sélecteurs de classes, eux-mêmes privilégiés face aux sélecteurs d'éléments.

Cette notion de priorité explique pourquoi les propriétés que vous tentez d'appliquer à un élément ne sont pas prises en compte : vous avez précédemment ciblé ce même élément à l'aide d'un sélecteur prioritaire.

Si tel est le cas, Firebug vous le signalera. Ajoutez du poids à votre sélecteur trop faible, ou employez le mot-clé `!important` (avec modération) pour donner des priorités à vos règles.

Donner le Layout

Si les étapes précédentes demeurent insuffisantes et si le problème de rendu n'est pas corrigé sur Internet Explorer, pensez au mécanisme *HasLayout*.

Commencez par vérifier si l'élément fautif possède le Layout. Si tel est le cas, tentez de l'enlever en annulant les propriétés CSS spécifiques au Layout et observez le résultat. Dans le cas contraire, tentez de conférer le Layout à l'élément qui pose problème, ou à son parent.

Dernier recours : le hack ou commentaire conditionnel

En dernier recours, et si l'erreur d'affichage ne se résout pas en attribuant le Layout aux éléments, il ne vous reste qu'une seule alternative : modifier les propriétés ou les valeurs, voire modifier les schémas de positionnement uniquement sur Internet Explorer, soit à l'aide de hacks, soit via une feuille de styles appelée par un commentaire conditionnel. Dans ce fichier de styles, vous placerez vos règles correctives pour le navigateur de Microsoft.