# 12

# Construire des pages Web dynamiques : donner du style à vos scripts

En 1996, Microsoft organisa une conférence à laquelle furent conviés un certain nombre d'auteurs et d'éditeurs. Un des responsables de l'entreprise présenta une projection montrant une page Web tout à fait ordinaire. Rien qui aurait pu justifier d'écrire ne serait-ce que quelques lignes à son sujet... tout du moins jusqu'à ce qu'il clique sur un titre, faisant apparaître juste au-dessous un paragraphe de texte jusque-là invisible. Une chose anodine aujourd'hui. Mais à l'époque, l'auditoire fut sidéré.

Ce fut la première présentation d'un concept qui allait ensuite être connu sous le nom de Dynamic HTML ou DHTML. La clé en était l'introduction par le W3C d'une nouvelle spécification, *Cascading Style Sheets* (*CSS*), qui venait s'ajouter au concept de Document Object Model (malgré l'absence d'un modèle vraiment universel à cette époque).

Les CSS permettent de définir l'aspect des éléments de page sans nécessiter des applications externes, des plug-ins ou un nombre excessif d'images. C'est également grâce aux CSS que nous pouvons séparer la présentation des pages de leur structure.

C'était toutefois grâce au DOM qu'il était possible d'accéder aux propriétés des feuilles de style depuis JavaScript. Nous pouvions ainsi modifier les propriétés des éléments même après le chargement de la page. Le DOM et les CSS formaient un moyen puissant de rendre les pages Web plus interactives que jamais.

Le seul problème était que chaque éditeur de navigateurs (Netscape Navigator et Microsoft Internet Explorer étant les plus populaires) implémentait un DOM différent, ce qui rendait DHTML difficile à mettre en œuvre. Bien que la version 4 de ces navigateurs ait été capable de produire des effets impressionnants, c'était au prix d'efforts de programmation importants. Chaque effet devait être codé différemment pour chaque navigateur et il fallait également penser à fournir une version pour les navigateurs dépourvus de DHTML. DHTML fut donc peu utilisé jusqu'à l'apparition des navigateurs les plus récents. Cette technologie est maintenant l'objet d'un intérêt important en raison de l'engouement pour Ajax (présenté aux chapitres 13 et 14).

**Info**

Comme nous l'avons dit au chapitre 11, nous avons développé un ensemble d'objets compatibles avec les différents navigateurs depuis 1998. Une variante moderne, ainsi que des exemples d'utilisation, peuvent être téléchargés depuis le site de l'auteur de ce livre, à l'adresse http://learningjavascript.info.

## DHTML : JavaScript, CSS et DOM

Les Cascading Style Sheets (CSS) ont connu des débuts difficiles. L'idée de placer la présentation des éléments de la page dans une spécification séparée était dans l'air depuis les débuts du Web, mais elle avait été écartée par les premiers développeurs de navigateurs. Il fallut attendre 1996 et la première apparition des CSS avec la diffusion des premiers navigateurs de quatrième génération pour que ce concept devienne une réalité. Ce n'était pas trop tôt, car les développeurs de pages Web étaient de plus en plus frustrés par les limitations de la présentation.

À cette époque, la plupart des pages étaient conçues sous forme de tableaux HTML, qui n'étaient d'ailleurs pas prévus pour cet usage mais pour la présentation de données. Un des problèmes posés par cette technique était que rien n'était affiché avant le chargement de toutes les images de la page. De plus, la gestion des tableaux était assez différente d'un navigateur à l'autre, ce qui compliquait beaucoup la mise au point. Si vous développiez alors des pages Web, vous utilisiez probablement la balise `font` ou pire, `blink`.

Les CSS apportèrent une solution propre et efficace. Grâce à cette technologie, vous pouviez initialiser et manipuler différentes catégories de propriétés déterminant la présentation des pages, comme la couleur de fond et de contenu, les bordures, la taille ou les marges internes et externes de chaque élément. Il manquait toutefois une fonctionnalité importante : la possibilité de positionner les éléments et de contrôler leur placement les uns par rapport aux autres, ainsi que leur visibilité. Il fallut attendre que Netscape et Microsoft collaborent sur un projet de spécification concernant le positionnement pour que ce problème trouve une solution. Cette spécification, d'abord connue sous le nom de CSS-P, devint par la suite CSS2.

**Info**

Dans ce chapitre, nous supposons que vous connaissez les feuilles de styles CSS et que vous savez comment les utiliser dans une page. Si ce n'est pas le cas, consultez un livre sur le sujet avant de lire ce chapitre. Nous vous recommandons en particulier *CSS*, de Eric A. Meyer (O'Reilly). Il existe également de nombreux tutoriels en ligne. Vous en trouverez notamment un en anglais à l'adresse http://www.w3schools.com/css/default.asp.

## La propriété style

Les valeurs des styles CSS des éléments peuvent être lues et modifiées en JavaScript par l'intermédiaire de leur propriété `style`. Ce concept a tout d'abord été proposé par Microsoft, puis adopté par le W3C qui l'a incorporé au module CSS DOM niveau 2. Dans ce modèle, tous les nœuds disposent d'un objet `style` accessible grâce à cette propriété, ce qui implique que n'importe quel élément peut voir sa présentation modifiée par JavaScript.

Pour modifier la présentation d'un élément en JavaScript, vous devez tout d'abord utiliser une des méthodes décrites aux chapitres 9 et 10 pour accéder à l'élément, puis modifier sa propriété `style` comme dans l'exemple suivant :

```
element.style.color="#fff";
```

Cette technique fonctionne avec tous les attributs de style CSS pour tous les objets XHTML. L'exemple 12-1 montre comment modifier différents attributs CSS d'un élément `div` obtenu grâce à la méthode `getElementById`.

**Exemple 12-1 – Modification des attributs de style d'un élément div**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>Modification des styles CSS</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<script type="text/javascript">
//<![CDATA[
document.onclick=changeElement;
function changeElement() {
  var div = document.getElementById("div1");
  div.style.backgroundColor="#00f";
  div.style.width="500px";
  div.style.color="#fff";
  div.style.height="200px";
```

Exemple 12-1 – Modification des attributs de style d'un élément div (*suite*)

```
  div.style.paddingLeft="50px";
  div.style.paddingTop="50px";
  div.style.fontFamily="Verdana";
  div.style.fontSize="2em";
  div.style.border="3px dashed #ff0";
  div.style.position="absolute";
  div.style.left="200px";
  div.style.top="100px";
  div.style.textDecoration="underline";
}

//]]>
</script>
</head>
<body>
<div id="div1">
Ceci est un élément DIV.
</div>
</body>
</html>
```

Notez dans cet exemple la convention employée pour les noms des styles CSS. Si le nom d'un style comporte un tiret, comme `border-color`, il est remplacé en JavaScript par une majuscule : `borderColor`. En dehors de ce détail, les noms sont les mêmes que dans les feuilles de styles. La figure 12-1 montre le résultat de la modification des styles une fois que l'utilisateur a cliqué dans la page.

Si la modification d'un style CSS est simple, la lecture de sa valeur l'est beaucoup moins. Si celle-ci n'est pas définie par un attribut `style` en ligne (dans le code HTML) ou en JavaScript, elle sera vide ou indéfinie, même si elle est déterminée dans une feuille de styles. Il est important de s'en souvenir car ce problème risque de vous coûter quelques cheveux lorsque vous travaillerez avec DHTML. Les styles utilisés pour l'affichage initial des objets sont internes au navigateur et sont basés sur une combinaison des valeurs indiquées dans la feuille de styles et de celles héritées des éléments parents.

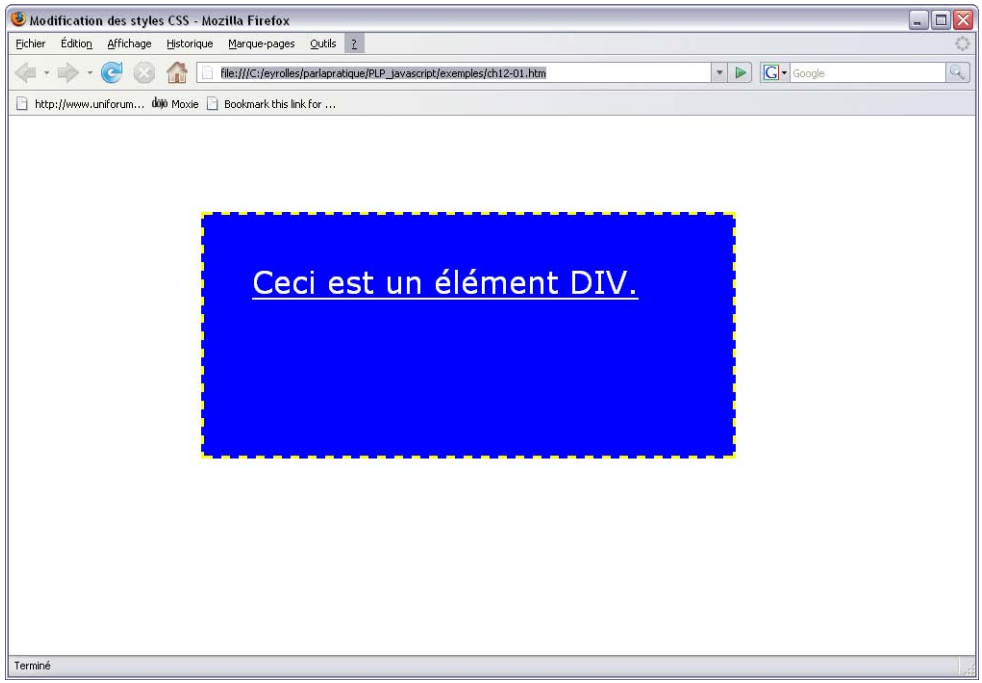



**Figure 12-1**
*Modification des styles CSS en JavaScript.*

> **Attention**
>
> Nous vous le répétons, à moins qu'un attribut de style ne soit déterminé par JavaScript ou au moyen d'un attribut `style` en ligne, sa valeur est vide ou indéfinie, même si elle a été fixée au moyen d'une feuille de styles.

Pour accéder aux valeurs de styles définies dans la feuille de styles, vous devez utiliser d'autres propriétés, spécifiques à chaque navigateur. Microsoft et Opera disposent de la propriété `currentStyle` pour chaque élément, alors que Firefox, Mozilla et Navigator emploient `window.getComputedStyle`. Malheureusement, ces propriétés fonctionnent de manières différentes.

Pour utiliser la méthode `getComputedStyle`, vous devez lui passer un attribut CSS qui utilise la même syntaxe que celle employée dans la feuille de styles. En revanche, la méthode `currentStyle` emploie la notation JavaScript. (Avec Safari, le choix d'une de ces deux méthodes n'a pas d'importance car aucune d'entre elles n'est supportée.)

L'exemple 12-2 montre les différentes façons de procéder. Il teste tout d'abord si `currentStyle` est supportée, puis effectue la même vérification pour `window.getComputedStyle`. Si aucune des deux propriétés n'est supportée, la fonction retourne `null`. Dans le cas contraire, la propriété est affichée avant et après sa modification.

**Exemple 12-2 – Lecture des valeurs de styles CSS**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>Styles</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<style type="text/css">
#div1 { background-color: #ff0 }
</style>
<script type="text/javascript">
//<![CDATA[
document.onclick=changeElement;
function getStyle(obj,jsprop,cssprop) {
   if (obj.currentStyle) {
       return obj.currentStyle[jsprop];
   } else if (window.getComputedStyle) {
      return
document.defaultView.getComputedStyle(obj,null).getPropertyValue(cssprop);   }
else {
     return null;
   }
}
function changeElement() {
    var obj = document.getElementById("div1");
    alert(obj.style.backgroundColor);
    alert(getStyle(obj,"backgroundColor","background-color"));
    obj.style.backgroundColor="#ff0000";
    alert(getStyle(obj,"backgroundColor","background-color"));
    alert(obj.style.backgroundColor);
}

//]]>
</script>
</head>
<body>
<div id="div1">
```

**Exemple 12-2 – Lecture des valeurs de styles CSS (*suite*)**

```
<p>Ceci est un élément DIV.</p>
</div>
</body>
</html>
```

Notez que la syntaxe employée pour lire la valeur de la propriété est `document.defaultView.getComputedStyle` au lieu de `window.getComputedStyle`. En effet, `document.defaultView` retourne l'objet DOM `AbstractView`, qui est l'interface de base dont dérivent toutes les vues. Elle peut correspondre à l'objet `window`, mais cela n'est pas garanti et peut varier d'un navigateur à l'autre, ou selon les versions. Il est donc préférable d'utiliser `document.defaultView`.

Même lorsque la propriété `style` est accessible, la valeur retournée dépend du navigateur. Par exemple, en ce qui concerne la couleur, Opera retourne la valeur au format hexadécimal :

```
#ff0000
```

alors que Firefox retourne :

```
RGB(255,0,0)
```

Vous devez alors effectuer la conversion si vous souhaitez obtenir un résultat cohérent.

La lecture des styles d'une page présente de nombreux challenges intéressants. Probablement trop pour que cela soit réellement divertissant. Une règle utile consiste à éviter de lire les données directement dans la page. Il est préférable de les stocker dans des variables du programme.

Les propriétés de style CSS sont regroupées par familles : polices, bordures, positionnement, affichage, etc. Dans la suite de ce chapitre, nous montrerons comment utiliser plusieurs de ces attributs en JavaScript. Prenez le temps d'étudier ces exemples et de les modifier pour bien comprendre leur fonctionnement.

**Info**

La lecture des valeurs de styles depuis la collection correspondant à la feuille de styles du document n'est pas traitée ici. Il s'agit d'une nouvelle collection qui ne fait pas partie du BOM. Cette approche permet de contourner certains problèmes de compatibilité et certaines difficultés mentionnées dans ce chapitre. Pour plus de détails et quelques exemples, reportez-vous à l'article *Modifying Styles*, de Steven Champeon, à l'adresse http://developer.apple.com/internet/webcontent/styles.html.

## Les polices et le texte

L'élément `font` a été un des premiers éléments de présentation HTML. C'est encore aujourd'hui un de ceux que vous trouverez (beaucoup trop) souvent dans les pages Web. Il n'est pas surprenant que les propriétés `font` et `text` aient une telle importance dans le développement des pages Web. Il y a peu d'attributs dont la modification puisse avoir autant d'effet sur la présentation des pages.

Notez la différence entre `font` et `text`. L'attribut `font` concerne la présentation des caractères proprement dits : la police, la taille, le type, etc. L'attribut `text` concerne le texte dans son ensemble, comme sa décoration ou son alignement.

### *Les propriétés font*

Il existe plusieurs attributs relatifs aux polices de caractères. Nous en donnons ci-après la liste, avec les notations équivalentes en JavaScript :

`font-family`

Cet attribut détermine la famille (police) de caractères, par exemple `Serif`, `Arial` ou `Verdana`. L'attribut JavaScript correspondant est `fontFamily`. Si le nom de la police contient des espaces, ils doivent être présents.

`font-size`

Cet attribut correspond à la taille des caractères. Plusieurs unités peuvent être employées. Avec `em` ou `pt` (par exemple `12pt` ou `2.5em`), les caractères sont dimensionnés en fonction des préférences de l'utilisateur. `px` (pixels) correspond à une taille fixe, quelles que soient les préférences configurées. Il est également possible d'utiliser les valeurs prédéfinies `xx-small`, `x-small`, `small`, `medium`, `large`, `x-large` et `xx-large`, ainsi que les valeurs relatives `smaller` ou `larger`, ou exprimées en pourcentage de la valeur de l'élément parent. Le nom de l'attribut JavaScript correspondant est `fontSize`.

`font-size-adjust`

En JavaScript, utilisez `fontSizeAdjust`. Cet attribut exprime le rapport entre la hauteur de la lettre `x` et la taille de la police.

`font-stretch`

L'attribut correspondant en JavaScript est `fontStretch`. Il correspond à une extension ou à une contraction des caractères. Les valeurs possibles sont `normal`, `wider`, `narrower`, `ultra-condensed`, `extra-condensed`, `condensed`, `semi-condensed`, `semi-expanded`, `expanded`, `ultra-expanded` ou `extra-expanded`.

`font-style`

`normal` (valeur par défaut), `italic` ou `oblique`. L'attribut JavaScript est `fontStyle`.

`font-variant`

`fontVariant` en JavaScript. Utilisez `small-caps` pour obtenir des petites capitales.

`font-weight`

Détermine les différentes valeurs de gras. Les valeurs reconnues sont `normal`, `bold`, `bolder`, `lighter` ou les valeurs numériques `100`, `200`, `300`, `400`, `500`, `600`, `700`, `800` ou `900`. Le nom JavaScript est `fontWeight`.

Nous avons vu dans l'exemple 12-1 que la modification de la police d'un élément concerne tous les éléments contenus par celui-ci, à moins que ces éléments ne soient modifiés explicitement. C'est cette modification en *cascade* qui donne leur nom aux CSS : *Cascading Style Sheets*. L'utilisation de JavaScript ne modifie pas cet effet.

Vous pouvez modifier plusieurs attributs de police de caractères en une seule instruction de la manière suivante :

```
div.style.font="italic small-caps 400 14px verdana";
```

L'attribut `font` est utilisé directement pour modifier les attributs `style`, `variant`, `weight`, `size` et `font-family`. La plupart des propriétés CSS offrent des raccourcis de ce type. Ils peuvent être employés en JavaScript de la même façon que dans les CSS. En CSS, toutes les valeurs se trouvent à droite du signe deux-points (:). En JavaScript, il suffit de les placer entre guillemets dans la partie droite de l'affectation.

## Les propriétés text

Dans ce chapitre, nous avons décidé de regrouper différents attributs qui déterminent la présentation du texte, bien que ceux-ci, contrairement à `font`, ne fassent pas partie d'une même famille. Les attributs de texte les plus couramment utilisés sont :

`color`

Détermine la couleur du texte.

`line-height`

`lineHeight` en JavaScript. Cet attribut correspond à l'espacement entre le bas d'une ligne et le haut de la suivante. Les valeurs possibles sont les mêmes que pour la taille des caractères. Vous pouvez également spécifier la valeur `normal`.

`text-decoration`

`textDecoration` en JavaScript. Les valeurs possibles sont `none`, `underline`, `overline` ou `line-through`. S'il vous plaît, évitez d'utiliser `blink` !

`text-indent`

`textIndent` en JavaScript. Détermine l'indentation de la première ligne.

`text-transform`

`textTransform` en JavaScript. Les valeurs possibles sont `none`, `capitalize` (pour mettre une majuscule à chaque mot), `uppercase` (majuscules) ou `lowercase` (minuscules).

`white-space`

`whiteSpace` en JavaScript. Détermine la façon de traiter les espaces et autres caractères non imprimables. Les différentes options sont `normal`, `pre` et `nowrap`.

`direction`

Indique le sens d'écriture du texte : `ltr` (de gauche à droite) ou `rtl` (de droite à gauche).

`text-align`

`textAlign` en JavaScript. Détermine l'alignement du texte : `left` (gauche), `right` (droite), `center` (centre) ou `justify` (aligné à gauche et à droite).

`word-spacing`

`wordSpacing` en JavaScript. Indique la quantité d'espace entre les mots. Vous pouvez indiquer une valeur ou utiliser `normal`.

Que pouvez-vous faire à l'aide de ces propriétés ? Vous pouvez par exemple élargir un bloc de texte pour le rendre plus visible ou mettre une partie en évidence pour attirer l'attention du lecteur. Dans l'exemple 12-3, un clic sur un des deux liens affiche le texte dans un grand bloc justifié ou le rétablit dans sa présentation d'origine.

**Exemple 12-3 – Modification d'un bloc de texte**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>Modification du texte/title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<script type="text/javascript">
//<![CDATA[
function makeMore() {
  var div = document.getElementById("div1");
  div.style.fontSize="larger";
  div.style.letterSpacing="10px";
  div.style.textAlign="justify";
```

**Exemple 12-3 – Modification d'un bloc de texte (*suite*)**

```
  div.style.textTransform="uppercase";
  div.style.fontSize="xx-large";
  div.style.fontWeight="900";
  div.style.lineHeight="40px";
}
function makeLess() {
   var div = document.getElementById("div1");
   div.style.fontSize="smaller";
   div.style.letterSpacing="normal";
   div.style.textAlign="left";
   div.style.textTransform="none";
   div.style.fontSize="medium";
   div.style.fontWeight="normal";
   div.style.lineHeight="normal";
}
//]]>
</script>
</head>
<body>
<p>
<a href="javascript:makeMore();">Plus gros</a> <a
href="javascript:makeLess();">Plus petit</a>
</p>
<div id="div1">
<p>
L'élément font a été un des premiers éléments de présentation HTML. C'est encore
aujourd'hui un de ceux que vous trouverez (beaucoup trop) souvent dans les pages
Web. Il n'est pas surprenant que les propriétés font et text aient une telle
importance dans le développement des pages Web. Il y a peu d'attributs dont la
modification puisse avoir autant d'effet sur la présentation des pages.
<p>
Notez la différence entre font et text. L'attribut font concerne la présentation
des caractères proprement dits : la police, la taille, le type, etc. L'attribut
text concerne le texte dans son ensemble, comme sa décoration ou son alignement.
</div>
</body>
</html>
```

Il est probable que vous ne souhaiterez pas agrandir le texte à ce point, mais cet exemple montre les transformations qui peuvent être effectuées en utilisant JavaScript et les CSS. Une

autre utilisation courante consiste à modifier la couleur de texte d'un élément de formulaire ou d'un bloc pour montrer qu'il est désactivé.

## Position et mouvement

Avant les CSS, il fallait utiliser des tableaux pour contrôler la mise en page du texte. Pour obtenir des effets d'animation, vous deviez employer des images GIF animées ou des plug-ins tels que Flash.

Netscape et Microsoft ont apporté conjointement une solution à ce problème avec la spécification CSS-P, ou CSS Positionning. Considérez la page comme un graphique avec des axes de coordonnées x et y. Avec la spécification CSS-P, chaque élément peut être positionné dans la page à l'aide de ses coordonnées. En ajoutant JavaScript, il devient facile de déplacer les éléments dans la page.

Les attributs CSS-P ont finalement été incorporés à la spécification CSS2. Les attributs de positionnement CSS3 sont les suivants :

`position`

La propriété `position` peut prendre cinq valeurs : `relative`, `absolute`, `static`, `inherit` ou `fixed`. Le positionnement `static` est utilisé par défaut pour la plupart des éléments. Cela signifie qu'ils font partie du flux normal des éléments de la page. La position d'un élément est déterminée par la présence des éléments précédents et elle influence à son tour la position des éléments suivants. La valeur `relative` indique qu'un élément est déplacé par rapport à sa position normale. Avec la valeur `absolute`, l'élément est sorti du flux normal pour être positionné de manière absolue. Il est ainsi possible de placer les éléments les uns sur les autres, en leur affectant la même position. `fixed` est identique à `absolute` à la différence que la référence est une vue (`viewport`). Pour la plupart des effets DHTML, vous utiliserez `relative` ou `absolute`.

`top`

Dans le système de coordonnées de la page Web, la valeur 0 de x correspond au haut du conteneur et elle augmente en descendant, que ce conteneur soit la page ou un autre élément. La propriété `top` détermine la position d'un élément à partir du haut de son conteneur.

`left`

Dans le système de coordonnées de la page Web, la valeur 0 de y correspond à la limite gauche du conteneur et elle augmente vers la droite, que ce conteneur soit la page ou un autre élément. La propriété `left` détermine la position d'un élément à partir de la limite gauche de son conteneur.

`bottom`

La propriété `bottom` correspond au bas du conteneur pour une valeur 0. Elle augmente en montant.

`right`

La propriété `right` correspond à la limite droite du conteneur pour une valeur 0. Elle augmente vers la droite.

`z-index`

La propriété `z-index` correspond à l'ordre d'empilement des éléments sur la page. Si deux éléments se chevauchent, leur visibilité est déterminée par deux caractéristiques. La première est l'ordre dans lequel ils apparaissent dans la page, le dernier élément étant placé au-dessus. Pour modifier l'ordre d'empilement, vous pouvez utiliser la propriété `z-index`. Sa valeur doit être entière, positive ou négative. Une valeur nulle correspond à la position par défaut. Une valeur négative repousse l'élément vers l'arrière-plan, alors qu'une valeur négative le déplace vers le premier plan.

L'attribut `display` a également une influence sur le positionnement des éléments, mais il sera traité dans la section *Affichage, visibilité et opacité*. L'attribut `float` est également lié au positionnement, mais son utilisation en DHTML est problématique et nous ne l'aborderons pas ici.

Les propriétés `top`, `right`, `bottom` et `left`, ainsi que `z-index`, n'ont une influence sur le positionnement que si `position` prend la valeur `absolute`. Des éléments peuvent être placés en dehors de la page en donnant des valeurs négatives à ces propriétés. Il est aussi possible de déplacer des éléments en fonction d'événements, par exemple un clic de souris.

Un des effets DHTML possibles consiste à faire apparaître un élément à partir d'un des côtés de la fenêtre. Cet effet peut être utile pour un tutoriel dans lequel des sujets doivent être affichés l'un après l'autre en fonction des clics de l'utilisateur ou des touches tapées au clavier.

L'exemple 12-4 montre trois éléments qui apparaissent depuis l'angle supérieur gauche de la page. Un timer est utilisé pour créer le mouvement. Il est réinitialisé chaque fois que la valeur de *x* (la coordonnée verticale) est supérieure à une valeur donnée (200 + une valeur × nombre d'éléments à afficher) pour créer une superposition. Dans leur position d'origine, les éléments sont invisibles car ils sont en dehors de la page, en haut et à gauche. (Un positionnement en bas ou à droite causerait l'affichage de barres de défilement.)

**Exemple 12-4 – Déplacement d'éléments**

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

Exemple 12-4 – Déplacement d'éléments (*suite*)

```
<title>Déplacement d'éléments</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<style type="text/css">
div { padding: 10px; }
#div1 { background-color: #00f;
        color: #fff;
        font-size: larger;
        position: absolute;
        width: 400px;
        height: 200px;
        left: -410px;
        top: -400px;
        }
#div2 { background-color: #ff0;
        color: #;
        font-size: larger;
        position: absolute;
        width: 400px;
        height: 200px;
        left: -410px;
        top: -400px;
        }
#div3 { background-color: #f00;
        color: #fff;
        font-size: larger;
        position: absolute;
        width: 400px;
        height: 200px;
        left: -410px;
        top: -400px;
        }
</style>
<script type="text/javascript">
//<![CDATA[
var element = ["div1","div2","div3"];
function next() {
   setTimeout("moveBlock()",1000);
}
var x = 0;
var y = 0;
```

**Exemple 12-4 – Déplacement d'éléments (*suite*)**

```
var elem = 0;
function moveBlock() {
   x+=20;
   y+=20;
   var obj = document.getElementById(element[elem]);
   obj.style.top = x + "px";
   obj.style.left = y + "px";
   if (x < (100 + elem * 60)) {
       setTimeout("moveBlock()", 100);
   } else {
      elem++;
      x = 0; y = 0;
   }
}
//]]>
</script>
</head>
<body>
<p>
<a href="javascript:next();">Prochaine diapo</a>
</p>
<div id="div1">
Avant les CSS, pour contrôler la mise en page du texte, il fallait utiliser des
tableaux. Pour obtenir des effets d'animation, vous deviez employer des images GIF
animées ou des plug-ins tels que Flash.</div>
<div id="div2">
Netscape et Microsoft ont apporté conjointement une solution à ce problème avec la
spécification CSS P, ou CSS Positionning. Considérez la page comme un graphique
avec des axes de coordonnées x et y. Avec la spécification CSS P, chaque élément
peut être positionné dans la page à l'aide de ses coordonnées. En ajoutant
JavaScript, il devient facile de déplacer les éléments dans la page.</div>
<div id="div3">
Un des effets DHTML possibles consiste à faire apparaître un élément à partir d'un
des côtés de la fenêtre. Cet effet peut être utile par exemple pour un tutoriel
dans lequel des sujets doivent être affichés l'un après l'autre en fonction des
clics de l'utilisateur ou des touches tapées au clavier.
</div>
</body>
</html>
```

La figure 12-2 montre le résultat affiché dans Firefox.

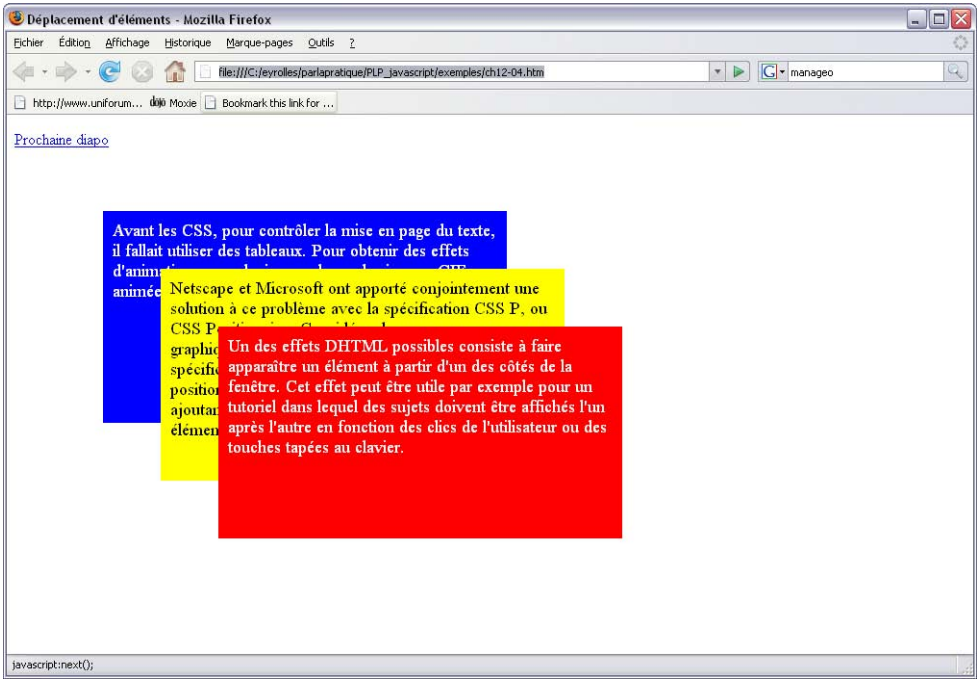



**Figure 12-2**
*Déplacement d'éléments.*

Pour rendre la page plus accessible, le lien peut être modifié afin d'ouvrir des pages affichant les informations contenues dans les trois blocs. Il est aussi possible de placer les trois blocs à leur position définitive en les masquant uniquement si JavaScript est activé, puis de les faire apparaître à la demande.

DHTML peut également être mis en œuvre pour implémenter une fonctionnalité essentielle : le glisser/déposer, présenté dans la section suivante.

## Le glisser/déposer

Le glisse/déposer est une des fonctionnalités qui a suscité le plus d'intérêt lors de la présentation de DHTML. Cette technique a été souvent employée, par exemple, pour manipuler le contenu des paniers sur les sites marchands. On a pu l'utiliser aussi pour développer des jeux interactifs.

Toutefois, avec le temps, l'intérêt pour le glisser/déposer dans les navigateurs s'est émoussé. Peu d'applications le mettent en œuvre et celles qui le font sont souvent irritantes. En effet, il

n'est pas toujours facile d'utiliser le glisser/déposer, surtout sur un portable équipé d'un *trackpad* ou avec un navigateur à lecture vocale.

L'application qui a ravivé l'intérêt pour le glisser/déposer est Google Maps. Cette application permet de déplacer une carte à l'écran en la faisant glisser. Il s'agit à notre avis de la première mise en œuvre utile et efficace de cette technique. Nous examinerons Google Maps ainsi que son API en détail au chapitre 13. Pour l'instant, nous allons développer une toute petite application qui utilise le glisser/déposer.

**Info**

Ce qui est fascinant avec Google Maps, c'est que cette application va chercher sur le serveur les éléments de carte nécessaires pour suivre les déplacements de l'utilisateur lorsqu'il fait glisser la carte. Ces éléments sont placés en mémoire cache pour un accès plus rapide. De ce fait, on a l'impression de ne jamais pouvoir atteindre les limites de la carte.

Dans l'exemple 12-5, un élément `div` est créé et une image est affichée dedans. En plus du glisser/déposer, une autre technique est mise en œuvre à l'aide de l'attribut `overflow` pour masquer la partie de la photo qui dépasse de l'élément `div`. Nous y reviendrons dans une prochaine section.

**Exemple 12-5 – L'effet Google Maps : faire glisser un objet dans son conteneur**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>Effet Google Maps</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<style type="text/css">
#div1 {
      overflow: hidden;
      position: absolute;
      top: 100px;
      left: 100px;
      border: 5px solid #000;
      width: 400px;
      height: 200px;
}
img {
     border: 1px solid #000;
    }
</style>
```

Exemple 12-5 – L'effet Google Maps : faire glisser un objet dans son conteneur (*suite*)

```
<script type="text/javascript">
//<![CDATA[
// Variables globales
var dragObject  = null;
var mouseOffset = null;
// Capture des événements souris
document.onmousemove = mouseMove;
document.onmouseup   = mouseUp;
// Création d'un point de souris
function mousePoint(x,y) {
   this.x = x;
   this.y = y;
}
// Trouver la position de la souris
function mousePosition(evnt){
  var x = parseInt(evnt.clientX);
  var y = parseInt(evnt.clientY);
  return new mousePoint(x,y);
}
// Lecture de la position de l'élément dans la page
function getMouseOffset(target, evnt){
   evnt = evnt || window.event;
   var mousePos  = mousePosition(evnt);
   var x = mousePos.x - target.offsetLeft;
   var y = mousePos.y - target.offsetTop;
   return new mousePoint(x,y);
}
// Désactivation du glisser
function mouseUp(evnt){
   dragObject = null;
}
// Capture du déplacement souris en cas de glisser uniquement
function mouseMove(evnt){
   if (!dragObject) return;
   evnt = evnt || window.event;
   var mousePos = mousePosition(evnt);
   // Détermination de la nouvelle position
   if(dragObject){
      dragObject.style.position = 'absolute';
      dragObject.style.top      = mousePos.y - mouseOffset.y + "px";
```

**Exemple 12-5 – L'effet Google Maps : faire glisser un objet dans son conteneur (*suite*)**

```
      dragObject.style.left     = mousePos.x - mouseOffset.x + "px";
      return false;
    }
}
// Activation du glisser pour un élément
function makeDraggable(item){
   if (item) {
      item = document.getElementById(item);
      item.onmousedown = function(evnt) {
                         dragObject  = this;
                         mouseOffset = getMouseOffset(this, evnt);
                         return false; };
   }
}
//]]>
</script>
</head>
<body onload="makeDraggable('img1');">
<div id="div1" >
<img id="img1" src="Image2.jpg" />
</div>
</body>
</html>
```

Cet exemple est le plus complexe que nous ayons présenté jusqu'ici et mérite une description détaillée :

- Deux objets de portée globale sont créés : `dragObject` et `mouseOffset`. Le premier est l'objet qui sera déplacé à l'aide de la souris. Le second représente la position du premier par rapport à son conteneur. Ici, le conteneur est la page. Nous interceptons également les événements `mousemove` et `mouseup` sur le document et nous les affectons aux gestionnaires d'événements `mouseMove` et `mouseUp`.
- L'objet suivant est `mousePoint`. Il s'agit simplement d'un enveloppeur pour les coordonnées *x* et *y* de la souris. Cet objet facilite le passage des coordonnées.
- Nous définissons ensuite la fonction `mousePosition`. Elle accède aux valeurs `clientX` et `clientY` de l'objet cible et retourne un objet `mousePoint` représentant les coordonnées *x* et *y* de la position par rapport à la zone utile de la fenêtre. La fonction `parseInt` vérifie que les valeurs retournées sont bien des nombres.

- La méthode `getMouseOffset` prend pour paramètre un objet `target` et un événement `event`. Une fois ce dernier normalisé par rapport aux différences entre les navigateurs, sa position est affectée à la variable `mousePos` grâce à la fonction `mousePosition`, après avoir soustrait les valeurs des propriétés `offsetLeft` et `offsetTop` des coordonnées *x* et *y*. Sans cette opération, l'objet se déplacerait avec la souris, mais il sauterait de façon saccadée à chaque clic pour s'ajuster à la position de la souris. Un nouveau point est ensuite retourné en passant les coordonnées à la fonction `mousePoint`.
- La fonction suivante, `mouseUp`, termine le glisser/déposer en donnant à `dragObject` la valeur `null`. La fonction `mouseMove` exécute l'essentiel des calculs nécessaires pour le déplacement. Si l'objet n'est pas déterminé, la fonction se termine immédiatement. Dans le cas contraire, la position normalisée de la souris est déterminée. La position prend la valeur `absolute` et ses propriétés `left` et `top` sont modifiées en tenant compte de l'ajustement correspondant à `mouseOffset`.
- La dernière fonction est `makeDraggable`. Elle fait en sorte que l'objet soit déplaçable en lui ajoutant une fonction gestionnaire de l'événement `mousedown`, qui affecte l'objet à `dragObject` et lit ses coordonnées.

Ce programme est assez long, mais beaucoup plus simple que ce qui était nécessaire avec les anciens navigateurs. En effet, les nouveaux navigateurs utilisent tous les mêmes propriétés pour la position des objets. Heureusement, car le glisser/déplacer est suffisamment complexe pour qu'il ne soit pas nécessaire d'y ajouter des difficultés inutiles. Google Maps ajoute un degré de sophistication en utilisant Ajax pour rafraîchir la carte de façon continue. Cette technique dépasse un peu le cadre de ce livre. Considérez-la comme un défi personnel !

## Contrôle de la taille et du détourage

La taille d'un élément est contrôlée par un ensemble de six attributs CSS. Les deux premiers, `width` et `height`, sont les plus courants et contrôlent la largeur et la hauteur des éléments. Les quatre autres, `min-height`, `min-width`, `max-height` et `max-width` sont pratiques pour manipuler des images, mais guère utilisés pour les effets dynamiques.

> **Info**
>
> En fait, les attributs `width` et `height` dépendent de différents attributs tels que `border`, `margin`, `padding` et `content`. Ensemble, ils définissent le modèle de boîte CSS associé à tous les éléments de type bloc, c'est-à-dire aux éléments qui provoquent une rupture de ligne avant et après eux. Pour plus d'informations sur ce modèle, reportez-vous à la page Box Model du site du W3C, à l'adresse http://www.w3.org/TR/REC-CSS2/box.htm.

Si les éléments contenus dans un bloc sont plus grands que celui-ci, le traitement de la partie excédentaire dépend de la valeur de l'attribut CSS `overflow`, qui peut prendre la valeur

`visible` (affichage des parties dépassant les limites du bloc), `hidden` (détourage, c'est-à-dire masquage des parties qui dépassent), `scroll` (masquage des parties excédentaires et affichage des barres de défilement) ou `auto` (détourage et affichage de barres de défilement uniquement si nécessaire).

**Info**

Pourquoi spécifier la hauteur d'un élément ? Après tout, si la hauteur n'est pas spécifiée, et si les parties dépassant ne sont pas masquées, l'élément est redimensionné automatiquement en fonction de son contenu.

Si vous disposez d'un contenu sur deux colonnes, il peut être nécessaire d'indiquer la taille des colonnes de façon que l'une d'elles ne soit pas beaucoup plus longue que l'autre.

## Détourage et contenu dynamique

Lorsque le contenu d'un élément est remplacé de manière dynamique, grâce à Ajax ou à un événement, la façon dont ce contenu remplit le bloc dans lequel il est placé peut changer de manière importante. Une possibilité consiste à donner à `overflow` la valeur `auto`. Si le contenu est plus grand que le bloc, des barres de défilement seront affichées. Dans l'exemple 12-6, deux blocs sont affichés. L'un contient une grande quantité de texte et l'autre beaucoup moins. Les dimensions des deux blocs sont déterminées de manière à permettre l'affichage du contenu. Un lien permet d'échanger le contenu des blocs. Dans les styles CSS, l'attribut `overflow` du second élément a la valeur `auto`.

**Exemple 12-6 – Traitement du dépassement du contenu par la propriété overflow**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>Overflow</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<style type="text/css">
#div1 { width: 700px; height: 200px }
#div2 { width: 600px; height: 100px; overflow: auto }
</style>
<script type="text/javascript">
//<![CDATA[
function switchContent() {
    var div1 = document.getElementById("div1").innerHTML;
    var div2 = document.getElementById("div2").innerHTML;
    document.getElementById("div1").innerHTML = div2;
```

**Exemple 12-6 – Traitement du dépassement du contenu par la propriété**

```
    document.getElementById("div2").innerHTML = div1;
}
//]]>
</script>
</head>
<body>
<p>
<a href="javascript:switchContent();">Echanger</a>
</p>
<div id="div1">
<p>
Avant les CSS, il fallait utiliser des tableaux pour contrôler la mise en page du
texte. Pour obtenir des effets d'animation, vous deviez employer des images GIF
animées, ou des plug-ins tels que Flash. Netscape et Microsoft ont apporté
conjointement une solution à ce problème avec la spécification CSS P, ou CSS
Positionning. Considérez la page comme un graphique avec des axes de coordonnées x
et y. Avec la spécification CSS P, chaque élément peut être positionné dans la page
à l'aide de ses coordonnées. En ajoutant JavaScript, il devient facile de déplacer
les éléments dans la page.
</p>
<p>
Un des effets DHTML possibles consiste à faire apparaître un élément à partir d'un
des côtés de la fenêtre. Cet effet peut être utile par exemple pour un tutoriel
dans lequel des sujets doivent être affichés l'un après l'autre en fonction des
clics de l'utilisateur ou des touches tapées au clavier.
</p>
</div>
<div id="div2">
<p>Texte court.</p>
</div>
</body>
</html>
```

Lorsque le contenu est échangé, le premier bloc affiche le texte court suivi d'un grand espace. Une seule méthode vous permet de modifier cet affichage : changer les dimensions du bloc `div`. Malheureusement, il n'est pas facile de déterminer la hauteur nécessaire.

Le deuxième bloc `div` affiche une barre de défilement à droite, ce qui permet de visualiser le contenu en totalité. Cette approche est meilleure car elle permet d'avoir des blocs de dimension et de position stables, dont le contenu est entièrement visible tout en évitant les grands espaces vides.

Une autre solution possible consiste à modifier les dimensions du bloc en utilisant les propriétés en lecture seule `offsetWidth` et `offsetHeight` pour déterminer la taille réelle du contenu. Il existe toutefois des différences entre les navigateurs en ce qui concerne ces propriétés. Internet Explorer inclut les bordures et les marges intérieures dans la taille du bloc, alors que Mozilla/Firefox ne considère que la taille du contenu.

> **Info**
>
> Il est également possible d'obtenir la taille d'un élément grâce à la méthode `getStyle` définie précédemment, en utilisant `width` et `height` au lieu de `backgroundColor`.

Bien que `width` et `height` permettent de contrôler la taille d'un élément, ces propriétés ne déterminent pas forcément sa partie visible. Celle-ci peut également être modifiée par le rectangle de détourage, ou *clippage*, ou encore *clipping*, associé à l'élément.

## Le rectangle de détourage

Le W3C donne la définition suivante de la notion de *clipping region* (zone de détourage) :

> ...définit la partie d'un élément dont le contenu est visible. Par défaut, la zone de détourage a la même taille et la même forme que la ou les boîte(s) contenant l'élément. Cette zone peut toutefois être modifiée grâce à la propriété `clip`. (*Visual Effects*, sur le site du W3C, à l'adresse http://www.w3.org/TR/REC-CSS2/visufx.html.)

La propriété CSS `clip` spécifie une forme et ses dimensions. Pour l'instant, la seule forme possible est le rectangle, désigné par `rect` et caractérisé par quatre dimensions : haut, droit, bas et gauche :

```
clip: rect(haut, droit, bas, gauche);
```

La zone de détourage détermine la partie visible de l'élément. Elle n'est applicable que si la valeur de la propriété `position` est `absolute`.

Si un élément a une largeur de 200 pixels et une hauteur de 300 pixels, une zone de détourage définie par `rect(0px,200px,300px,0px)` ne masque aucune partie, sauf si cet élément possède une bordure ou des marges modifiant ses dimensions. Une zone de détourage définie par `rect(10px,190px,290px,10px)` masque dix pixels de chaque côté. Notez qu'une augmentation des valeurs des côtés haut et gauche assortie d'une diminution des valeurs des côtés droit et bas entraîne un détourage.

Du point de vue de DHTML, le détourage peut être employé pour créer un effet de défilement, qui peut être associé ou non au déplacement de l'élément. Il permet également de créer un effet « accordéon » très à la mode qui sera décrit au chapitre 14.

L'exemple 12-7 montre une mise en application simple du détourage pour créer une animation sous la forme d'un élément déroulant. L'élément peut être affiché ou masqué en cliquant sur son titre. Un timer est employé pour animer l'effet. Vous pouvez également afficher ou masquer simplement l'élément en ignorant le timer.

**Exemple 12-7 – Animation d'un élément déroulant mettant en œuvre le détourage et un timer**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>Elément déroulant</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<style type="text/css">
#data1     {
           position: absolute;
           top: 100px; left: 100px;
           padding: 0;
           width: 200px;
           height: 200px;
           background-color: #ff0;
           clip: rect(0px,200px,200px,0px);
           }
#data1 h3 {
          height: 20px;
          margin: 0; padding: 5px;
          font-size: smaller;
          background-color: #006;
          color: #fff;
          }
#contained {
           margin: 10px
           }

</style>
<script type="text/javascript">
//<![CDATA[
var bottom = 200;
var hidden = false;
var obj = null;
function clipItem() {
  obj = document.getElementById("data1");
```

**Exemple 12-7 – Animation d'un élément déroulant mettant en œuvre le détourage et un timer (*suite*)**

```
  if (hidden) {
       showItem();
   } else {
       hideItem();
   }
}
function hideItem() {
   bottom-=10;
   var clip = "rect(0px,200px," + bottom + "px,0px)";
   obj.style.clip = clip;
   if (bottom <= 30) {
      hidden=true;
   } else {
      setTimeout("hideItem()",10);
   }
}
function showItem() {
   bottom+=10;
   var clip = "rect(0px,200px," + bottom + "px,0px)";
   obj.style.clip=clip;
   if (bottom >= 200) {
      hidden=false;
   } else {
      setTimeout("showItem()",10);
   }
}
//]]>
</script>
</head>
<body>
<div id="data1">
<h3 onclick="clipItem();">Cliquez pour afficher ou masquer</h3>
<div id="contained">
Ceci est le texte contenu dans le bloc div, qui peut être affiché ou masqué.
</div>
</div>
</body>
</html>
```

Notez que la valeur de détourage n'est pas lue directement dans la propriété `style`, mais qu'une variable globale est utilisée. Cette approche est toujours préférable avec les animations DHTML. La lecture d'une variable est toujours plus rapide que celle de la propriété d'un objet, surtout lorsque des problèmes de compatibilité entre navigateurs doivent être pris en compte.

## Affichage, visibilité et opacité

Des éléments d'une page Web peuvent être complètement transparents et invisibles, mais affecter tout de même la disposition des autres éléments. En effet, en CSS, la visibilité d'un élément n'est pas automatiquement liée à son affichage.

Un élément peut être caché en donnant à sa propriété `visibility` la valeur `hidden`. Pour le rendre visible de nouveau, il suffit de donner à cette propriété la valeur `visible`. La valeur `inherit` peut également être utilisée afin que l'élément se comporte comme son conteneur.

Comme nous l'avons montré au chapitre 11, l'opacité d'un élément peut aussi être modifiée jusqu'à le rendre complètement transparent, et donc invisible. L'élément reste toutefois présent dans la page, comme lorsque `visibility` prend la valeur `hidden`.

Si la valeur de la propriété `display` est `none`, l'élément n'est plus visible et n'influence plus la disposition des autres éléments de la page. Pour le rendre visible, plusieurs options sont possibles. Vous pouvez donner à `display` la valeur `block` afin de l'afficher avec une rupture de ligne avant et après. Si vous ne souhaitez pas lui donner ce comportement, vous pouvez utiliser la valeur `inline` afin de l'afficher sans rupture de ligne.

Il est également possible d'utiliser des valeurs représentant le comportement standard de divers éléments HTML. Parmi ces valeurs, citons notamment `inline-block`, `table`, `table-cell`, `list-item`, `compact`, `run-in`, etc. Cet attribut permet d'obtenir des résultats intéressants. Faites vos propres expériences afin de vous familiariser avec son comportement.

### *Utiliser les outils adaptés*

Étant donné le nombre de possibilités pour masquer et afficher des éléments, il peut sembler difficile de déterminer la plus adaptée.

Dans le cas d'un élément en position absolue qui doit être masqué ou affiché en fonction d'un événement tel qu'un clic de souris, utilisez la propriété `visibility`. Elle est d'une utilisation facile et ce type d' élément est simplement retiré de la page lorsqu'il n'est pas visible.

Si le contenu masqué doit pousser les autres éléments vers le bas lors de l'affichage, comme dans le cas d'un accordéon, utilisez la propriété `display` en alternant entre les valeurs `none` et `block`. Utilisez également `display` pour masquer et afficher les champs de formulaires en fonction des entrées de l'utilisateur.

Pour créer un effet d'effacement par transparence ou pour montrer qu'un élément est désactivé, employez la propriété `opacity`. Vous pouvez ainsi rendre un objet totalement transparent, mais ce sera généralement après un effet d'animation. La propriété `opacity` peut également être employée pour obtenir un effet de transition, comme dans le diaporama du chapitre 11.

**Info**

Les effets visuels doivent être accompagnés de texte pour que les utilisateurs de navigateurs non visuels, ainsi que les personnes aux capacités visuelles limitées, puissent bénéficier des mêmes informations. Ne comptez jamais uniquement sur l'effet visuel pour informer l'utilisateur.

## Information permanente

Les meilleurs sites fournissent une assistance permanente chaque fois qu'une information est demandée à l'utilisateur. Par exemple, lorsque celui-ci doit entrer son nom, la page affiche une explication concernant le respect de la vie privée et l'usage qui sera fait de ces données.

Il est possible d'afficher des *info-bulles* grâce à l'attribut `title` des liens, mais la quantité d'informations qui peut y être placée est réduite. Une autre solution consiste à afficher une fenêtre de type *pop-up*, qui peut contenir une plus grande quantité d'informations, par exemple la description de différentes options. Mais pour les cas intermédiaires, il serait utile de pouvoir afficher ces informations directement dans la page.

Mais les formulaires occupent beaucoup d'espace et y ajouter une grand quantité de texte peut rendre la page moins lisible. Une solution possible consiste à placer le texte dans la page, mais à l'afficher uniquement lorsqu'un événement donné se produit.

Il s'agit là d'un des effets DHTML les plus utiles et les plus faciles à réaliser. L'exemple 12-8 crée une page contenant deux éléments de formulaire possédant chacun un bloc de texte d'aide. Lorsque l'utilisateur clique sur l'étiquette d'un élément, le texte est affiché. Si un autre bloc d'aide était affiché préalablement, il est tout d'abord masqué.

**Exemple 12-8 – Utilisation de textes d'aide masqués**

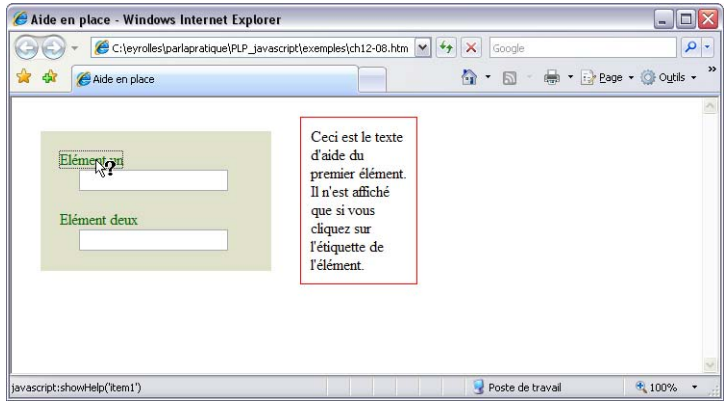
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>Aide en place</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<style type="text/css">

.help { position: absolute;
        left: 300px;
        top: 20px;
```

**Exemple 12-8 – Utilisation de textes d'aide masqués (*suite*)**

```
        visibility: hidden;
        width: 100px;
        padding: 10px;
        border: 1px solid #f00;
      }
form { margin: 20px; background-color: #DFE1CB;
       padding: 20px; width: 200px }
form a {color: #060; text-decoration: none }
form a:hover {cursor : help}
</style>
<script type="text/javascript">
//<![CDATA[
var item = null;
function showHelp(newItem) {
   if (item) {
       item.style.visibility='hidden';
   }
   item = document.getElementById(newItem);
   item.style.visibility='visible';
}
//]]>
</script>
</head>
<body>
<form>
<label><a href="javascript:showHelp('item1')" alt="get help">Elément un</a></
label>
<input type="text"><br /><br />
<label><a href="javascript:showHelp('item2')" alt="get help">Elément deux</a></
label>
<input type="text">
</form>
<div id="item1" class="help">
Ceci est le texte d'aide du premier élément. Il n'est affiché que si vous cliquez
sur l'étiquette de l'élément.
</div>
<div id="item2" class="help">
Ceci est le texte d'aide du second élément. Il n'est affiché que si vous cliquez
sur l'étiquette de l'élément.
</div>
</body>
</html>
```

Nous avons ajouté quelques éléments de style CSS pour rendre l'exemple plus présentable. Le formulaire possède un fond de couleur, les blocs d'aide sont encadrés en rouge et, lorsque le pointeur de la souris est placé sur une étiquette, il affiche l'icône `help`, représentant une flèche et un point d'interrogation, ou parfois un point d'interrogation seul. C'est un moyen simple d'informer l'utilisateur, tout comme l'attribut `alt` qui affiche « Cliquez pour obtenir de l'aide. » La figure 12-3 montre le résultat obtenu.

**Figure 12-3**
*Le système d'aide.*




## Des formulaires à géométrie variable

La division d'un formulaire en plusieurs pages est pénible, mais une page comportant un trop grand nombre d'éléments est difficilement lisible.

Par ailleurs, l'*édition en place* est une technique de plus en plus appréciée : l'utilisateur peut cliquer sur un titre ou une étiquette et les données correspondantes deviennent modifiables grâce à un champ de formulaire.

Dans ces deux situations, des formulaires rétractables peuvent être mis en œuvre. Il s'agit de formulaires dont certaines sections sont masquées. Elles ne sont affichées que lorsqu'un événement particulier se produit. Le reste du formulaire est alors décalé vers le bas pour faire de la place aux éléments nouvellement affichés.

Google, Flickr et de nombreux autres sites emploient cette technique, ce qui n'est pas surprenant lorsque l'on sait à quel point elle est facile à mettre en œuvre. Si JavaScript n'est pas activé, les éléments masqués ne seront jamais affichés. Il est possible d'utiliser un menu pour afficher le formulaire complet dans une autre page ou pour afficher tous les éléments grâce à la balise `noscript`.

L'exemple 12-9 illustre ce type de formulaire. Il est composé d'un empilement de blocs. Un clic sur une étiquette masque le bloc correspondant s'il est affiché ou l'affiche dans le cas contraire. Pour les navigateurs dans lesquels JavaScript n'est pas activé, les étiquettes sont

placées dans des liens qui affichent une version statique du formulaire. Si JavaScript est actif, ce comportement est annulé par le retour de la valeur **false** par le gestionnaire d'événement **onclick**. Vous pouvez vérifiez cela si vous désactivez JavaScript : lorsque vous cliquez sur un lien, l'URL de la page est modifié pour refléter la cible (**#nom** ou **#adresse**).

**Exemple 12-9 – Un formulaire à zones rétractables**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>Formulaire à zones rétractables</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<style type="text/css">
.label { background-color: #003; width: 400px; border-right: 1px solid #fff;
         padding: 10px; margin: 0 20px; color: #fff; text-align: center;
         border-bottom: 1px solid #fff;}
.label a { color: #fff }
.elements { background-color: #CCD9FF; margin: 0 20px; padding: 10px;
            width: 400px; display: none}
</style>
<script type="text/javascript">
//<![CDATA[
window.onload=setup;
function setup() {
   document.getElementById('one').style.display='none';
   document.getElementById('two').style.display='none';
}
function show(newItem) {
   var item = document.getElementById(newItem);
   if (item.style.display=='none') {
       item.style.display='block';
   } else {
       item.style.display='none';
   }
}
//]]>
</script>
</head>
<body>
<form action="GET">
<div class="label" onclick="show('one')">
<a href="#nom" onclick="return false">Nom</a>
```

**Exemple 12-9 – Un formulaire à zones rétractables (*suite*)**

```
</div>
<div class="elements" id="one">
<label>Prénom :</label><br /><input type="text" name="prenom" /><br /><br />
<label>Nom :</label><br /><input type="text" name="nom" /><br /><br />
</div>
<div class="label" onclick="show('two')">
<a href="#adresse" onclick="return false">Adresse</a>
</div>
<div class="elements" id="two">
<label>Rue :</label><br /><input type="text" name="rue" /><br /><br />
<label>Code postal :</label><br /><input type="text" name="codepostal" /><br /><br
/>
<label>Ville :</label><br /><input type="text" name="ville" /><br /><br />
</div>
</form>
<p>Autres données.</p>
</body>
</html>
```

Ce type de fonctionnalité est très utile. Sa mise en œuvre est facile, son effet est impressionnant et il est aisé d'implémenter une solution alternative qui n'utilise pas JavaScript.

Nous avons simplement effleuré la surface de ce qu'il est possible de réaliser avec JavaScript et les CSS. C'est un bon point de départ. Le chapitre 13 sera consacré aux aspects fondamentaux d'Ajax. Nous verrons ensuite comment Ajax et DHTML peuvent être associés pour construire des applications réellement impressionnantes.

## Questions

1. Vous tentez de lire la couleur de texte d'un élément en JavaScript grâce à la propriété `obj.style.color`, mais aucune valeur n'est retournée. Vous savez que la couleur a été spécifiée dans la feuille de styles. Pourquoi n'obtenez-vous pas sa valeur et comment pouvez-vous remédier à ce problème ?
2. Comment pouvez-vous modifier un bloc de texte `div` pour afficher le texte en rouge, taille 14 pt et interligne 16 pt ?
3. Dans quel cas l'opération précédente peut-elle échouer ?
4. Citez deux façons de faire disparaître un bloc.

5. Si le glisser/déposer n'est pas une technique efficace pour la gestion d'un panier sur un site de vente, quel effet DHTML pouvez-vous utiliser ?

Les réponses se trouvent dans l'annexe.