5

Les médias en HTML5 : l'audio et la vidéo

Aujourd'hui, le Web est véritablement devenu multimédia. L'audio et la vidéo sont partie intégrante du contenu dans lequel nous naviguons quotidiennement. Grâce à la croissance constante de la bande passante et des technologies de compression, c'est maintenant monnaie courante de regarder la télévision sur un périphérique mobile, ou encore un film en streaming sur un téléviseur grâce à des services comme Apple TV et Google TV. De plus, la plupart des chaînes de télévision offrent des contenus gratuits sur leurs sites web. Et nous ne parlerons pas de la popularité de YouTube ou de sites de partage vidéo similaires.

Malgré la présence importante de contenu multimédia, il n'y avait, avant le HTML5, aucun standard ouvert pour sa transmission sur le Web. En fait, elle était assurée (et elle l'est toujours) par des plugins tiers, tels que QuickTime, Windows Media Player, Flash Player et Real Player.

Au départ, la situation était chaotique. La disponibilité inégale des différents plugins poussait ceux qui souhaitaient publier des vidéos sur leur site web à les proposer sous plusieurs formats que les lecteurs les plus courants pouvaient lire. Cela nécessitait un encodage de la même vidéo avec différents codecs avant la publication sur le Web. Un codec vidéo est un logiciel qui permet la compression et/ou la décompression de la vidéo numérique. En général, la compression emploie des méthodes avec "perte de données", ce qui signifie que pendant l'opération, certaines informations sont perdues. Vous pouvez en lire davantage sur les codecs vidéo sur Wikipédia : **http://en.wikipedia.org/wiki/Video_codec**.

On a assisté à une véritable guerre entre les différents plugins et, entre 2005 et 2006, l'un d'eux a finalement émergé : Adobe Flash Player. Pendant des années, il a été très populaire (installé sur 90 % des machines connectées à Internet). Avant d'être racheté par Adobe, Macromedia avait inséré le format vidéo dans Flash Player, et il n'a pas fallu longtemps pour que les grandes sociétés du Web l'adoptent. Cela a eu pour effet que tout le monde pouvait virtuellement lire les vidéos fondées sur Flash.

Flash Video est un format de fichier conteneur, utilisé pour transmettre des vidéos sur Internet avec Adobe Flash Player, version 6 à 11. Ce contenu peut également être intégré dans des fichiers *swf*.

Deux formats de vidéo sont reconnus comme formats Flash Video : *flv* et *f4v*. Les données audio et vidéo contenues dans les fichiers *flv* sont encodées de la même manière que dans les fichiers *swf*. Les deux formats, actuellement développés par Adobe Systems, sont pris en charge par Adobe Flash Player. *flv* a été originellement développé par Macromedia. Il s'est rapidement imposé comme le format de choix pour l'intégration de vidéos sur le Web. Parmi ses utilisateurs connus, on trouve YouTube, Hulu, Google Vidéos, Yahoo! Vidéo, Metacafe, Reuters.com, et beaucoup d'autres fournisseurs.

Bien que les formats Flash Video et Flash Player soient très populaires et largement pris en charge, Apple a choisi de les ignorer pour ses périphériques (iPhone, iPad et iPod Touch). En revanche, ils prennent en charge à la place le HTML5.

Si vous souhaitez faire des recherches sur ce sujet, vous pouvez commencer par consulter Thoughts on Flash*, rédigé par Apple disponible ici* **http://www.apple.com/hotnews/thoughts-on-flash/***, et la réponse d'Adobe ici :* **blogs.wsj.com/digits/2010/04/29/live-blogging-the-journals-interviewwith-adobe-ceo/**.

Pour plus d'information sur Flash Video, consultez **http://fr.wikipedia.org/wiki/Flash_Video**.

Lors de cette période de grand développement, le HTML offrait pour sa part seulement `<object>` et `<embed>`, deux balises pour intégrer du contenu utilisable par un plugin externe à une page web.

Le HTML5 offre un standard natif ouvert, afin que du contenu multimédia puisse être proposé. Les spécifications du langage étant encore en cours de définition, ces nouvelles balises ne sont prises en charge pour le moment que par les versions des navigateurs listées dans le Tableau 5.1.

Tableau 5.1. Navigateurs prenant en charge les éléments *audio* et *video* du HTML5

Google	*Internet Explorer*	*Safari*	*Firefox*	*Opera*	*iPhone*	*Android*
3 +	9 +	3 +	3.5 +	10.5 +	1 +	2 +

Le problème actuel n'est pas l'adoption des nouvelles balises HTML5 puisque ce dernier définit un standard de codecs à utiliser (voir Figure 5.1). Le problème est que Ogg Theora, H.264 et VP8/WebM sont actuellement les seuls formats pris en charge par l'élément `video`. Lequel d'entre eux sera retenu de manière universelle dans le futur ? Nous n'avons pas encore la réponse.

Actuellement, un vaste débat a lieu à ce sujet et à propos des différents points de vue associés. Le plus discuté est la position de Google, qui a acquis On2 technologies et a rendu *open-source* le codec vidéo VP8/WebM. En janvier 2011, Google a annoncé qu'il abandonnait la prise en charge du codec H.264 pour Chrome, alors que Apple et Microsoft continuent de le supporter.

On2 Technologies produit les codecs vidéo suivant : VP3, VP4, VP5, VP6, VP7 et VP8. Ogg Theora est dérivé du codec propriétaire VP3 mis dans le domaine public par On2 Technologies.

Pour résumer, une nouvelle bataille a commencé pour établir le prochain codec vidéo. Comme toujours, nous ne pouvons qu'attendre de voir comment cela va évoluer.

Browser	Latest stable release version date	Native video format support			
		Ogg Theora	H.264	VP8 (WebM)	Others
Internet Explorer	8.0 (March 19, 2009; 21 months ago)	No[note 1]	9.0[19]	No[note 2]	No[21]
Mozilla Firefox[22]	3.6.13 (December 9, 2010; 39 days ago)	3.5[23]	No[note 3]	4.0[25][26]	No
Google Chrome	8.0.552.237 (January 12, 2011; 5 days ago)	3.0[27][28]	Yes (3.0)[29] / No (deprecated)[30]	6.0[31][32]	No[33]
Chromium	N/A	r18297[34]	Depends[35]	r47759[36]	No[33]
Safari	5.0.3 (November 18, 2010; 60 days ago)	No[note 4]	3.1[37][38]	No	Depends[note 5]
Opera	11.00 (Build 1156) (December 16, 2010; 32 days ago)	10.50[40]	Depends[note 6]	10.60[42][43]	Depends[note 6]
Konqueror	4.5.5 (7 January 2011; 10 days ago)[44][45]	4.4[46]	No[note 7]	Depends[note 7]	Depends[note 7]

Figure 5.1
Le statut des codecs vidéo sur le Web et leur prise en charge par les différents navigateurs.
Source : **http://en.wikipedia.org/wiki/HTML5_video**.

Nous présentons ici des solutions pour utiliser les nouveaux marquages HTML5 afin de pouvoir travailler avec le contenu multimédia audio et vidéo.

Solution 5.1 – Intégrer de la vidéo dans une page web

Publier une vidéo dans une page web est devenu, avec le HTML5, une opération vraiment simple. L'ère où nous devions faire appel à des plugins tiers pour rendre une vidéo accessible dans une page HTML est sur le point de s'achever.

Les codes comme le suivant pourraient bien devenir un vague souvenir :

```
<object width="640" height="480">
<param name="movie" value="your_video.swf">
<embed src="your_video.swf" width="500" height="500">
</embed>
</object>
```

Pour regarder une vidéo, l'utilisateur n'aura plus à télécharger un plugin additionnel ni à mettre à jour ceux qui sont déjà installés. Il n'aura plus, non plus, à subir les dysfonctionnements dus à l'instabilité de certains plugins tiers.

Voyons comment les choses changent en HTML5.

Tour d'horizon

Avec l'introduction de la nouvelle balise `<video>`, tout ce que nous devons faire est de déclarer le marquage dans la page web, autrement dit d'indiquer la vidéo qui doit être chargée, et le navigateur fera le reste (en supposant qu'il prenne en charge l'élément `video`) :

```
<video src="your_video.ogg" />
```

L'attribut `src` contient l'adresse de la ressource média (audio ou vidéo) qui doit s'afficher dans la page. Dans l'exemple de code précédent, nous lui avons demandé de charger la vidéo au format Ogg Theora.

Même si ce code seul suffit à rendre la vidéo utilisable, on a cependant intérêt à spécifier les dimensions du conteneur vidéo avec les attributs `width` et `height` :

```
<video width="640" height="360" src="your_video.mp4" />
```

Si on ne le fait pas, le navigateur utilisera les dimensions d'origine de la ressource vidéo.

Les autres propriétés prises en charge par la balise `video` sont les suivantes :

- `preload`. Indique au navigateur de précharger le contenu vidéo alors que la page se charge. Ainsi, l'utilisateur n'aura pas à attendre que la vidéo se charge lorsqu'il la lira.
- `autoplay`. Indique au navigateur de jouer la vidéo automatiquement dès qu'elle est disponible. Faites attention avec cet attribut car vous ne pouvez pas être toujours sûr que l'utilisateur voudra voir cette vidéo. Cela est particulièrement vrai s'il est connecté *via* un périphérique mobile où la bande passante est plus onéreuse.
- `loop`. Réexécute la vidéo dès qu'elle est achevée.
- `controls`. Si cette propriété est activée, cela indique au navigateur d'afficher le groupe d'éléments de contrôle intégrés comme lecture, stop, pause et volume.
- `poster`. Indique un fichier image que le navigateur peut afficher lorsque la donnée vidéo n'est pas disponible.

Mise en œuvre

L'exemple de code qui suit montre comment importer puis afficher une vidéo dans une page web :

```
<!DOCTYPE html>
<html>
<head>
<title>
Solution 5-1: Intégrer de la vidéo dans une page web
</title>
</head>
<body>
<h1>Comtaste's Showreel</h1>
<video width="640" height="360"
src="comtaste_showreel.mp4" autoplay />
</body>
</html>
```

Si vous ouvrez cet exemple dans un navigateur, la vidéo sera exécutée immédiatement et occupera un espace de 640 pixels de large et de 800 pixels de haut (voir Figure 5.2).

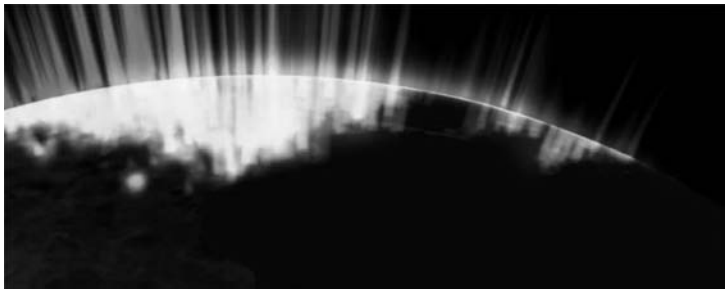


Figure 5.2
La vidéo se lira automatiquement dans la page web.

Si vous souhaitez essayer cette solution pour charger une vidéo distante, utilisez le code suivant : `<video width="640" height="360" src="http://www.youtube.com/demo/google_main.mp4" autoplay />`.

Si l'utilisateur ouvre le fichier HTML dans un navigateur qui ne prend pas en charge la balise `video`, une page blanche s'affiche. Une bonne pratique consiste, par conséquent, à proposer un contenu alternatif dans le cas où cela arrive. Utilisez alors l'attribut `poster` pour afficher une image qui pourrait être, par exemple, une capture d'image de la vidéo. Elle peut être soit locale, soit provenir d'un autre endroit sur le Web.

Ajoutons cet attribut à la balise `video` :

```
<video width="640" height="360" src="comtaste_showreel.mp4" autoplay poster="../
➥ img/Figure_5_3.png"> Video n'est pas pris en charge par ce navigateur! </video>
```

Si la vidéo ne se charge pas, le navigateur affichera l'image spécifiée par l'attribut `poster` (voir Figure 5.3).

Figure 5.3
Le navigateur affiche l'image dans l'attribut poster.

Si l'attribut `poster` n'est pas spécifié et si le navigateur ne peut pas charger la vidéo, c'est la première image de la vidéo qui s'affichera par défaut.

Conseil d'expert

Quelques problèmes apparaissent avec la prise en charge de la vidéo HTML5 des périphériques Apple iOS (iPhone, iPod Touch et iPad) :

- Si vous utilisez l'attribut `poster`, iOS ignorera l'élément `video`. Apple a annoncé que ce problème était résolu dans iOS 4.0.
- iOS ne prend en charge que le format H.264. Si vous utilisez la balise `<source>` (voir la solution suivante), il reconnaîtra seulement le premier format vidéo.
- Les périphériques Android ne prennent pas en charge les éléments de contrôle natifs du navigateur et les ignorent. De même, le système d'exploitation est un peu confus en ce qui concerne l'attribut `type` qui sert à spécifier le conteneur `video`.

Solution 5.2 – Détecter la prise en charge de la vidéo dans les différents navigateurs

Nous avons déjà évoqué le large choix de formats vidéo disponibles sur le Web. Les premières spécifications du HTML5 ont établi que la prise en charge des éléments video par un navigateur devait se baser sur deux formats : Ogg Vorbis (pour l'audio) et Ogg Theora (pour la vidéo). Cette déclaration a provoqué une certaine agitation parmi les grands acteurs impliqués, tels que Apple et Nokia, à tel point que le WSC - *Web Standards Curriculum* - a retiré toute référence aux formats audio et vidéo des spécifications.

Ce choix a évidemment entraîné un certain chaos. MPEG4 (H.264), Ogg Theora, AVIS et VP8 WebM sont tous prêts à mener une guerre pour devenir le standard du futur. Les navigateurs eux-mêmes ont commencé à prendre parti (voir Figure 5.4).

- Opera : Ogg Theora et WebM dans le futur.
- Chrome : Ogg Theora et WebM (Google a récemment annoncé l'abandon de H.264).
- Firefox : Ogg Theora et WebM dans le futur.
- Safari : H.264.
- Internet Explorer : H.264 et WebM.

Figure 5.4
La prise en charge des conteneurs vidéo par les navigateurs.

	CHROME	EXPLORER	FIREFOX	OPERA	SAFARI
H.264- MP4	No Support	9	No Support	No Support	3.2 +
Ogg Theora - OMG	3 +	No Support	3.5 +	10.5 +	No Support
VP8 - WebM	6 +	9	4	10.6 +	No Support

On comprend donc facilement pourquoi il est nécessaire, étant donné la situation actuelle, d'opérer des vérifications de navigateurs lorsque la page se charge, afin de choisir les formats vidéo.

Tour d'horizon

Plusieurs techniques permettent de vérifier le format vidéo pris en charge par le navigateur. La solution ci-après s'appuie sur la bibliothèque JavaScript Modernizr, disponible sur **http://www.modernizr.com/**, dont nous avons déjà parlé au Chapitre 1.

Nous pouvons également nous servir des fonctions natives de cette bibliothèque vidéo pour vérifier la prise en charge de la balise `video` et des codecs. En voici un exemple :

```
if (Modernizr.video) {
if (Modernizr.video.webm) {
  // prend en charge WebM
}
else if (Modernizr.video.ogg){
  // prend en charge Ogg Theora + Vorbis
} else if (Modernizr.video.h264){
 // prend en charge H.264 vidéo + AAC audio }
}
```

La bibliothèque exécute cette vérification en testant la propriété `canPlayType` de l'élément `video`.

Le navigateur retourne alors l'une de ces trois valeurs :

- Une chaîne vide : il ne prend pas l'élément en charge.
- La chaîne `maybe` : il n'est pas sûr de prendre l'élément en charge.
- La chaîne `probably` : il prend en charge la combinaison du conteneur et du codec.

Mise en œuvre

Pour créer une page HTML qui vérifie la prise en charge de la balise `video` puis charge le bon code, choisissez un mécanisme analogue à celui de la solution précédente, mais présentant certaines différences.

Voici un exemple de code complet qui utilise la bibliothèque Modernizr afin de détecter la prise en charge de la vidéo sur les différents navigateurs :

```
<html>
<head>
<title>
Solution 5-2: Détecter la prise en charge de la vidéo dans les différents navigateurs
</title>
<script type="text/JavaScript" src="js/modernizr-1.0.min.js"></script>
<style type="text/css">
body { background: #f7f7f7; color: black; text-align: center; }
.video #no-video,
.no-video #video { display: none; }
</style>
</head>
<body>
  <h1>Comtaste's Showreel</h1>
```

```
  <div id="video">
  <h1> Ceci est un élément vidéo HTML5 </h1>
  <video width="640" height="360"
  src="comtaste_showreel.mp4" autoplay poster="../img/Figure_5_3.png" />
  </div>
</body>
</html>
```

Au moment de l'écriture de cet ouvrage, la dernière version de la bibliothèque Modernizr est la 1.7, mais cette solution fonctionne également avec des versions plus anciennes.

Dans ce code, nous avons créé deux `DIV` : une, qui contient le code pour les navigateurs qui prennent en charge la balise `<video>` (et dont l'`id` est `#video`), et une autre pour les autres navigateurs (dont l'`id` est `#no-video`). Nous avons également inséré un bloc de style contenant les déclarations CSS nécessaires pour cacher la `DIV` qui n'est pas utilisée, grâce à la commande `display:none`.

Dans la première `DIV` (celle dont l'`id` est `#video`), nous avons inséré la balise `<video>`. La seconde, elle, utilise une technique écrite par Kroc Camen, qui ne nécessite pas de JavaScript mais deux vidéos encodées – un fichier *ogg* et un fichier *MP4* – et qui se fonde sur la supposition que, si la vidéo HTML5 n'est pas prise en charge, alors Adobe Flash sera utilisé à la place.

Cette approche est compatible avec le HTML4, le HTML5 (marquage valide) et le XHTML1. Cela fonctionne également dans le cadre d'une application XHTML+XML. Vous pouvez en savoir davantage en lisant l'article "Video for Everybody" disponible à cette adresse : **http://camendesign.com/code/video_for_everybody**.

Dans la seconde `DIV`, celle qui se chargera si le navigateur ne prend pas en charge la balise `video`, insérez le code suivant :

```
<!-- alternative Flash: -->
<object width="640" height="360" type="application/x-shockwave-flash"data=
➥ "__FLASH__.SWF">
<!-- Firefox utilise l'attribut 'data' ci-dessus, et IE/Safari utilisent
➥l'attribut param ci-après
<param name="movie" value="comtaste_showreel.SWF" />
<param name="flashvars" value="controlbar=over&amp;image= comtaste_showreel.
➥JPG&amp;file=comtaste_showreel.mp4" />
<!-- image alternative -->
<img src="comtaste_showreel.JPG" width="640" height="360"
title="pas de fonctionnalité vidéo, téléchargez la vidéo ci-après" />
</object>
```

Conseil d'expert

Il existe d'autres solutions pour travailler avec des vidéos sur le Web. L'une d'elles, qui rencontre un franc succès, est le projet HTML5media disponible sur **https://github.com/etianen/html5media**.

Ce projet détecte le support à la fois de la balise `video` et des formats de vidéo grâce à un lecteur multimédia HTML5. Il prend en charge les formats H.264(MP4) et Ogg Theora.

Si le navigateur ignore la balise `video` HTML5, il lance le lecteur Adobe Flash Player pour procurer les mêmes fonctions que la vidéo originale. C'est pourquoi nous utilisons la bibliothèque JavaScript FlowPlayer(**http://www.flowplayer.org**).

Pour activer la balise HTML5 `video` dans n'importe quel navigateur, vous devez appeler la bibliothèque Jquery et le script dans la balise `head` du document comme ceci :

```
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.min.js">
➥</script>
<script src="http://html5media.googlecode.com/svn/trunk/src/jquery.html5media.
➥min.js">
</script>
```

Vous pouvez ensuite intégrer de la vidéo dans une page HTML avec le code suivant :

```
<video src="video.mp4" autoplay autobuffer></video>
```

D'autres bibliothèques existent sur le même sujet : **http://mediaelementjs.com/** et **http://videojs.com/**.

Solution 5.3 – Créer un contrôleur vidéo personnalisé

Tout élément multimédia, qu'il soit audio ou vidéo, doit proposer à l'utilisateur la possibilité d'interagir avec le contenu en cliquant sur les boutons classiques de lecture, stop, pause et volume. Le HTML5 dispose d'un attribut qui les affiche nativement sur un contrôleur vidéo.

Il est cependant possible (et souvent préférable) de créer des boutons de contrôle personnalisés afin qu'ils correspondent au graphisme du site web sur lequel la vidéo est publiée.

Les éléments audio et vidéo font partie des DOM (*Document Object Model*) HTML5, qui proposent une API puissante et facile d'utilisation pour contrôler la lecture vidéo. Dans cette solution, nous allons voir comment ajouter des éléments de contrôle personnalisés à une vidéo.

Tour d'horizon

Intégrer des contrôles dans un contenu vidéo est une opération facile. En fait, un attribut `controls` existe pour la balise `<video>` et il tire avantage des contrôles intégrés du navigateur. Vous devez spécifier l'attribut dans la balise afin que les contrôles s'affichent sur la vidéo.

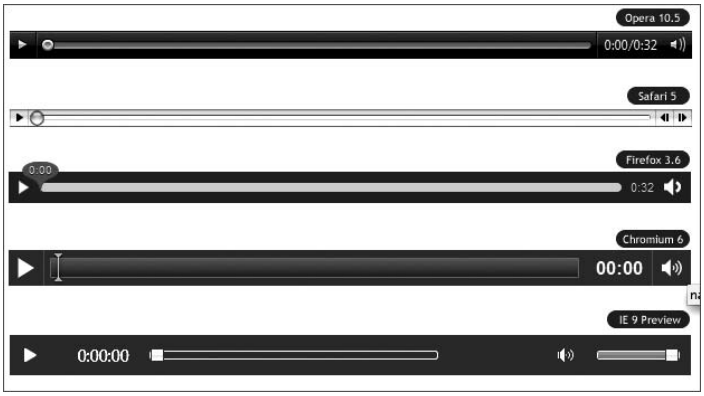
```
<video width="640" height="360" src="comtaste_showreel.mp4" controls />
```

Chaque navigateur utilisera ses propres éléments graphiques pour le design des contrôles vidéo. Par exemple, à la Figure 5.5, on voit les éléments graphiques utilisés pour les contrôles vidéo dans Safari.

Figure 5.5
Les contrôles vidéo affichés par Safari.

Cette divergence va probablement ennuyer les designers qui veulent maîtriser l'apparence de leurs contrôles vidéo afin de les faire correspondre au graphisme du site web dans lequel ils se trouvent. On voit à la Figure 5.6 les éléments graphiques utilisés pour les contrôles vidéo dans les autres principaux navigateurs.

Figure 5.6
La façon dont les principaux navigateurs affichent les contrôles vidéo.



Néanmoins, vous pouvez créer des contrôles vidéo personnalisés, utilisant vos propres éléments graphiques et choisir la fonction à leur allouer. Vous pouvez, par exemple, écrire une fonctionnalité pour synchroniser les sous-titres d'une vidéo et les activer avec des contrôleurs vidéo ; vous pouvez encore écrire une fonctionnalité permettant à l'utilisateur de sauter d'un signet à un autre pour voir la partie qui l'intéresse le plus.

Vous avez le choix de faire selon votre volonté puisque c'est vous qui écrivez les fonctions du contrôleur vidéo en JavaScript. Pour cela, vous devez utiliser les attributs média HTML5 et les événements DOM que vous pouvez écouter, tels que la progression du chargement, la lecture du média, la mise en pause et la fin de la lecture. Par exemple, voici les attributs pour les fonctionnalités de lecture d'une vidéo :

- `media.paused`. Renvoie `true` si la lecture est en pause, et `false` autrement.
- `media.ended`. Renvoie `true` si la lecture a atteint la fin du média.
- `media.defaultPlaybackRate [ = value ]`. Renvoie la vitesse de lecture par défaut lorsque l'utilisateur n'est pas en train d'accélérer en avant ou en arrière sur la source du média.

- `Media. playbackRate [ = value ]`. Renvoie la vitesse de lecture ; 1.0 est la vitesse normale.
- `Media.played`. Renvoie un objet `TimeRanges` qui représente la durée de la ressource média que le navigateur a lue.
- `Media.play()`. Définit l'attribut `media.paused` sur `false`, ce qui entraîne le chargement de la ressource média et le début de sa lecture si nécessaire. Si la lecture est finie, elle recommencera depuis l'origine.
- `Media.pause()`. Définit l'attribut `media.paused` sur `true`, ce qui charge la ressource média, si nécessaire.
- `Media.volume`. Récupère ou définit la valeur du volume de la bande-son de la vidéo. C'est un nombre flottant compris entre 0.0 (silencieux) et 1.0 (le plus fort).
- `Media.muted`. Coupe le son.
- `Media.currentTime`. Retourne la position actuelle de lecture en secondes sous la forme d'un nombre flottant.

Pour obtenir une liste des attributs média et des événements, référez-vous à la page suivante : **http://www.whatwg/.org/specs/web-apps/current-work/multipage/video.html**.

En utilisant ces méthodes, et quelques lignes de JavaScript, vous pouvez aisément créer un contrôleur vidéo simple :

```
if (video.paused){
  video.play();
}
```

Cette condition vérifie si la vidéo est en pause ou non. Dans l'affirmative, elle l'exécute avec la méthode `play()`. Par contre, si la vidéo est finie, on revient au début à l'aide de la propriété `currentTime`, et on peut la relire :

```
if (video.paused){
  video.play();
} else if (video.ended){
  video.currentTime=0;
  video.play();
}
```

En fait, cela est l'affaire de quelques lignes de code. Voyons comment créer un exemple complet.

Mise en œuvre

Commencez par créer deux blocs `DIV`. Le premier contiendra l'élément `video`, et le second, les contrôles vidéo.

Créez ensuite un fichier et ajoutez la balise `<video>` dans un bloc `DIV` auquel vous assignerez un `id` équivalent à `video_container`. L'`id` est important car vous pourrez y accéder par le JavaScript et le CSS.

Voici le fragment de code :

```
<div id="video_container">
<video width="320" height="176" src="comtaste_showreel.mp4" />
</div>
```

Il charge une vidéo au format MPEG et lui donne les dimensions de 320 × 176 pixels.

Insérez un second bloc `DIV`, qui contiendra les contrôles vidéo de lecture, de mise en pause, de volume, d'audio muet et le temps. Pour cela, utilisez la balise `<button>` pour les boutons Lecture, Pause et Muet, et un slider pour contrôler le volume. Les informations sur la durée de lecture sont contenues dans un `label` :

```
<div id="video_controller">
<button id="btn_play"> Lecture </button>
<button id="btn_pause"> Pause </button>
<input type="range" min="0" max="1" step="0.1" id="volume">
<button id="btn_mute"> Muet </button>
<label id="time">-:--:--</label>
</div>
```

Il est également important, dans ce cas, de porter attention aux noms que l'on donne aux attributs `id` des balises car vous en aurez besoin ultérieurement pour référencer les objets en JavaScript.

Le résultat final de cette interface utilisateur de la page web est visible à la Figure 5.7.

Figure 5.7
L'élément vidéo et son contrôleur vidéo personnalisé.

Vous devez maintenant programmer le fonctionnement du contrôleur vidéo. Insérez un bloc de script et commencez avec un gestionnaire d'événement au chargement de la page. Dans ce dernier, définissez une variable globale qui contiendra l'élément `video`, et une variable locale qui contiendra la référence aux contrôleurs vidéo :

```
<script type="text/JavaScript">
var video;
window.onload = function(){
  video = document.getElementsByTagName("video")[0];
  var btn_play = document.getElementById("btn_play");
  var btn_pause = document.getElementById("btn_pause");
  var btn_mute = document.getElementById("btn_mute");
  var btn_volume = document.getElementById('volume');
```

Nous avons utilisé la méthode `getElementByTagName` afin d'accéder à l'élément `video`, et la méthode `getElementById` pour accéder au premier élément avec l'`id` spécifié pour référencer les

autres contrôles. Ainsi, nous pouvons parvenir, par exemple, à l'élément `video` se référant à la variable `video` globale sans avoir à rappeler la méthode `getElementById`.

Pour que les boutons exécutent les opérations sur la vidéo, vous devez associer leur événement de clic à un gestionnaire d'événement (qui est en fait une méthode), où vous écrirez les opérations que le bouton de contrôle doit exécuter. Pour associer l'événement d'un élément à une méthode, nous utilisons `addEventListener()`.

Ajoutez d'abord les lignes de code suivantes (nous sommes toujours en train d'écrire le code dans l'événement `onload` de l'objet `window`) :

```
btn_play.addEventListener('click', doPlay, false);
btn_pause.addEventListener('click', doPause, false);
btn_mute.addEventListener('click', doMute, false);
```

Trois gestionnaires d'événement ont été associés au clic sur les boutons : `doPlay()`, `doPause()` et `doMute()`.

Pour finir, avant de fermer la déclaration de l'événement `onload` de la page, ajoutez un gestionnaire d'événement pour gérer le volume de l'élément `video`, qui fonctionnera sur l'événement `change` du slider.

Ajoutez le code suivant :

```
btn_volume.value = video.volume;
btn_volume.addEventListener('change',function(e) {
myVol= e.target.value;
video.volume=myVol;
if (myVol==0) {
video.muted = true;
} else {
video.muted = false;
}
return false;
}, true);
};
```

Analysons le code que nous venons d'écrire. Dès que vous ouvrirez la page web, l'élément `video` héritera du volume défini dans les paramètres utilisateur. C'est pourquoi vous avez dû définir la valeur du slider comme valeur actuelle du volume la vidéo :

```
btn_volume.value = video.volume;
```

Ainsi, vous êtes sûr que, lorsque l'utilisateur modifiera le volume de la vidéo, il le fera à partir de sa valeur initiale réelle.

Ensuite, nous avons créé un gestionnaire d'événement pour l'événement `change`, qui survient lorsque l'utilisateur déplace le curseur du slider, autrement dit, qu'il change le volume de l'élément `video`, qui sera équivalent à la valeur du slider :

```
myVol= e.target.value;
video.volume=myVol;
```

Nous exécutons une condition pour vérifier si le volume est égal à 0. Ce paramètre définit l'attribut `muted` de la vidéo, laquelle n'est pas muette automatiquement lors du changement de volume :

```
if (myVol==0) {
video.muted = true;
} else {
video.muted = false;
}
```

Avec ces quelques lignes de code, vous venez d'écrire la fonction de contrôle de volume pour l'élément `video`.

Nous pouvons maintenant commencer à écrire les gestionnaires d'événement pour les boutons. Ajoutez le code suivant :

```
function doPlay(){
if (video.paused){
video.play();
} else if (video.ended)
{
video.currentTime=0;
video.play();
};
};
function doPause(){
if (video.play){
video.pause();
};
};
function doMute(){
document.getElementById('volume').value = 0;
video.muted = true;
};
```

Avec les méthodes `play()`, `pause()` et `muted`, nous avons fourni les fonctions pour les boutons du contrôleur vidéo.

Vous pouvez maintenant sauvegarder le fichier et l'exécuter dans un navigateur. Vous verrez que les boutons Lecture, Pause et Muet, fonctionnent, et vous pourrez également changer le volume de la vidéo en déplaçant le curseur du slider.

Voici le code complet de cet exemple :

```
<!DOCTYPE html>
<html>
<head>
<title>
Solution 5-3: Créer un contrôleur vidéo personnalisé
</title>
<script type="text/JavaScript">
var video;
```

```
window.onload = function(){
  video = document.getElementsByTagName("video")[0];
  var btn_play = document.getElementById("btn_play");
  var btn_pause = document.getElementById("btn_pause");
  var btn_mute = document.getElementById("btn_mute");
  var btn_volume = document.getElementById('volume');
  btn_play.addEventListener('click', doPlay, false);
  btn_pause.addEventListener('click', doPause, false);
  btn_mute.addEventListener('click', doMute, false);
  btn_volume.value = video.volume;
  btn_volume.addEventListener('change',function(e) {
  myVol= e.target.value;
  video.volume=myVol;
  if (myVol==0) {
    video.muted = true;
  } else {
    video.muted = false;
  }
  return false;
  }, true);
};
function doPlay(){
  if (video.paused){
    video.play();
  } else if (video.ended){
    video.currentTime=0;
    video.play();
  };
};

function doPause(){
  if (video.play){
    video.pause();
  };
};

function doMute(){
  document.getElementById('volume').value = 0;
  video.muted = true;
};
</script>
</head>
<body>
<div id="video_container">
<video width="320" height="176" src="comtaste_showreel.mp4" />
</div>
<div id="video_controller">
```

```
<button id="btn_play"> Lecture </button>
<button id="btn_pause"> Pause </button>
<input type="range" min="0" max="1" step="0.1" id="volume">
<button id="btn_mute"> Muet </button>
</div>
</body>
</html>
```

Conseil d'expert

L'élément `video` révèle un événement qui permet de déterminer la durée restante de la vidéo en exécution.

L'événement `timeupdate` est exécuté comme une partie normale de la lecture lorsque la position actuelle de la lecture change.

Nous allons ajouter un marquage `label` qui contiendra la durée de la vidéo pendant qu'elle est lue.

Insérez le code suivant dans le bloc `DIV` :

```
<button id="btn_play"> Lecture </button>
<button id="btn_pause"> Pause </button>
<input type="range" min="0" max="1" step="0.1" id="volume">
<button id="btn_mute"> Muet </button>
<label id="time">-:--:--</label>
</div>
```

Nous allons intégrer les valeurs de durée d'exécution de la vidéo en heures, minutes et secondes dans le label en utilisant du JavaScript.

Ajoutez un gestionnaire d'événement pour l'événement `timeupdate` de l'élément `video`. Insérez la ligne de code suivante dans la fonction de l'événement `onload` de l'objet `window` :

```
video.addEventListener('timeupdate', updateTime, false);
```

Puis déclarez la méthode `updateTime` qui écrira la valeur heures/minutes/secondes dans le `<label>` que nous avons créé, dans la vue de la page web :

```
function updateTime(){
  var sec= video.currentTime;
  var h = Math.floor(sec/3600);
  sec=sec%3600;
  var min =Math.floor(sec/60);
  sec = Math.floor(sec%60);
  if (sec.toString().length < 2) sec="0"+sec;
  if (min.toString().length < 2) min="0"+min;
  document.getElementById('time').innerHTML = h+":"+min+":"+sec;
}
```

Cette fonction a uniquement pour tâche de scinder la valeur `currentTime`, exprimée en secondes, en chaînes séparées heures/minutes/secondes, qu'elle écrira ensuite dans le `label` ayant `time` pour `id` grâce à la méthode `innerHTML` :

```
document.getElementById('time').innerHTML = h+":"+min+":"+sec;
```

Ajoutez ensuite un peu de CSS pour agrémenter le champ `label` :

```
<style type="text/css">
#video_container {
  margin: 0;
  padding: 0;
}
#time {
  margin: 0;
  padding: 5px;
  width: 350px;
  font-family: Helvetica, Arial, sans-serif
  font-size: .7em;
  color: #000000;
  background-color: #ccc;
}
</style>
```

Vous pouvez voir le résultat final à la Figure 5.8.

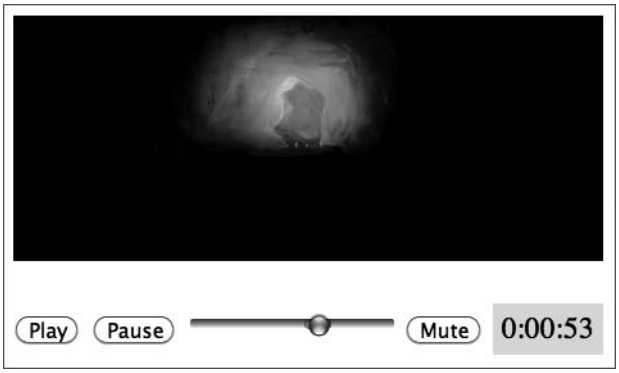


Figure 5.8
L'élément video et ses contrôleurs vidéo en état de marche.

Solution 5.4 – Précharger une vidéo

Il existe deux méthodes principales pour distribuer des médias sur Internet :

- le flux continu ou *streaming* ;
- le téléchargement progressif.

Évoquer les différences existant entre ces deux méthodes dépasse le cadre de ce livre. Tout ce que nous avons besoin de savoir est que le streaming utilise un serveur et un protocole pour permettre à l'utilisateur de voir n'importe quelle partie de la vidéo sans qu'il ait à attendre que la vidéo soit chargée en totalité, et ce tout au long de la vidéo.

La seconde méthode s'appuie sur le protocole standard HTTP pour transmettre le fichier. Dans ce cas, l'utilisateur ne peut "sauter" vers une partie de la vidéo qui n'a pas encore été chargée.