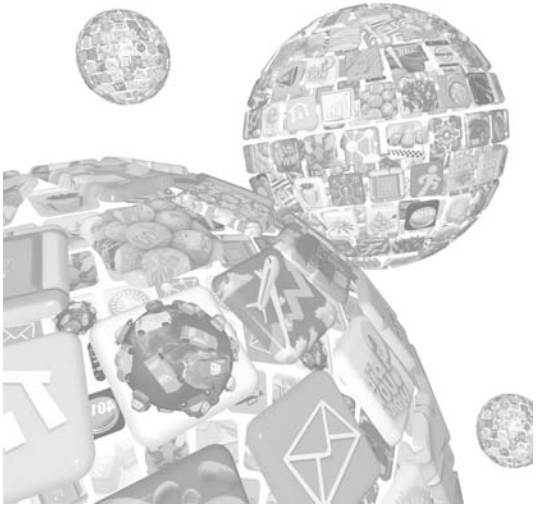


8

Contenu audio et vidéo

Au sommaire de ce chapitre

- Incorporer du contenu vidéo
- Incorporer du contenu audio
- Rester raisonnable
- Prendre le contrôle sur le contenu
- Ajouter des sous-titres et des chapitres
- Réponses aux différents problèmes
- Contenu multimédia pour l'iPhone et l'iPad

Certes, Safari Mobile ne prend pas en charge le contenu Flash, mais iOS est une plateforme multimédia extraordinaire qui bénéficie de la qualité audio et vidéo de l'iPod. Le fait que l'application YouTube soit fournie par défaut avec le système d'exploitation est symptomatique de l'importance accordée au multimédia par Apple et les sites web tiers. Les sociétés qui optimisent leur site pour l'iPhone et l'iPad sont toujours plus nombreuses, par exemple Vimeo (voir Figure 8.1 page suivante), et le contenu Adobe Flash classique est remplacé par des vidéos au format H.264 soutenu par Apple. YouTube fournit même une nouvelle version mobile de son site web qui, grâce aux nouvelles fonctionnalités de HTML5, surpasse l'application native intégrée à l'iPhone.

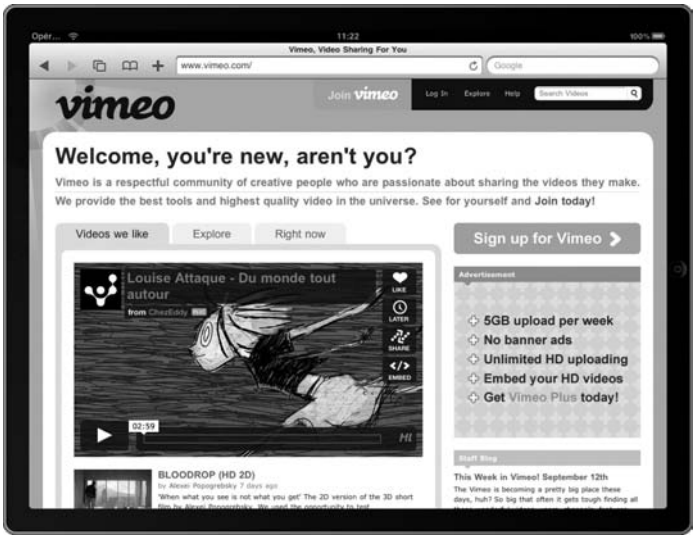


Figure 8.1
L'application web de Vimeo optimisée pour l'iPad.

De nombreux signes montrent que les acteurs du Web ont de plus en plus tendance à se tourner vers des formats alternatifs propres au Web ; Adobe a même ajouté la reconnaissance de H.264 dans Flash Player. Cette tendance fait notamment suite à l'excellente prise en charge des derniers standards web par les navigateurs mobiles, comme Safari Mobile, avec pour conséquence une gestion efficace du contenu multimédia. Vous allez le voir, Apple conseille non plus d'employer la balise `<embed>` dans Safari mais d'adopter les balises HTML5 `<audio>` et `<video>`, dont les API permettent de contrôler de nombreux aspects de la lecture et de collecter des informations en temps réel sur l'état du contenu au travers d'un large éventail d'événements.

Pour illustrer ce chapitre, nous utilisons le superbe film d'animation Big Buck Bunny *(sous license Creative Common, © Blender Foundation). Il est disponible en téléchargement sur le site* **www.bigbuckbunny.org** *dans de nombreux formats, dont celui de l'iPhone.*

Avant HTML5, il n'existait aucun standard pour incorporer du contenu multimédia dans une page web et les navigateurs avaient généralement recours à des plug-in externes, qui demandaient parfois une installation de la part de l'utilisateur, chose impossible sous Safari Mobile. Tout cela est obsolète, et il ne tient qu'à vous de vous lancer.

Incorporer du contenu vidéo

Le contenu multimédia le plus répandu aujourd'hui sur le Web sont probablement les vidéos. Même si vous ne produisez pas vos propres clips, rien ne vous empêche d'en inclure dans votre site. Par exemple, le code suivant exploite YouTube :

```
<a href="http://www.youtube.com/v/IDENTIFIANT_DE_LA_VIDEO">
    <img src="metropolis.jpg" alt="Affiche Metropolis">
</a>
```

`IDENTIFIANT_DE_LA_VIDEO` est la valeur du paramètre `v` qui se trouve dans toutes les URL des vidéos YouTube. Toutefois, incorporer de cette manière un contenu vidéo déclenche le lancement de l'application YouTube et, une fois le visionnage terminé, l'utilisateur reste dans l'application.

Puisque vous proposez votre propre application web, vous préféreriez évidemment que l'utilisateur y reste aussi longtemps que possible et qu'il bénéficie d'une expérience aussi proche que possible de celle d'une application native. Grâce à HTML5, la lecture d'une vidéo directement depuis le navigateur demande seulement l'utilisation de la nouvelle balise `<video>`. Elle constitue une solution simple et performante pour inclure du contenu vidéo, semblable par certains aspects à l'ancienne balise `<embed>` avec laquelle elle partage plusieurs attributs :

```
<video src="mavideo.m4v" poster="apercu.png" id="mavideo"
    width="320" height="240"></video>
```

À l'instar de la balise `<canvas>`, la balise `<video>` possède une largeur par défaut de 300 pixels et une hauteur de 150 pixels. Pour modifier ces dimensions, servez-vous des attributs `width` et `height`. Si le rapport largeur/hauteur de la vidéo est différent de celui de la zone définie, des bandes noires sont affichées de manière à remplir l'espace inoccupé et la vidéo est redimensionnée en conservant le rapport d'origine. Malheureusement, il est impossible de modifier la couleur des bandes, par exemple en appliquant une couleur d'arrière-plan comme vous auriez pu l'imaginer.

Par défaut, la balise `<video>` est un élément en ligne et doit être fermée comme dans notre exemple. Dans le cas contraire, le code qui suit sera considéré comme du contenu de repli. Par ailleurs, la balise `<video>` peut contenir d'autres balises, comme `<object>` ou `<embed>`, ou d'autres éléments qui permettent d'affiner la définition du type de média et d'ajouter des fonctionnalités supplémentaires à la lecture.

Obtenir des informations sur la vidéo

Il peut arriver que vous ne connaissiez pas le rapport largeur/hauteur initial de la vidéo. Dans ce cas, les propriétés de l'objet `HTMLVideoElement` seront très utiles, par exemple pour ajuster la taille du conteneur. Elles peuvent également servir si vous proposez un format par défaut de la vidéo, mais donnez la possibilité d'opter pour d'autres formats selon la rapidité de la connexion de l'appareil de l'utilisateur.

```
var video = document.getElementById("mavideo");
video.addEventListener("loadedmetadata", adjustVideo, false);

function adjustVideo(event) {
    var video = event.target;

    /* Taille intrinsèque (en lecture seule). */
    var vw = video.videoWidth;
    var vh = video.videoHeight;

    /* Taille de l'élément. */
    var ew = video.offsetWidth;  // Ou video.width.
```

```
        var eh = video.offsetHeight; // Ou video.height.

        /* Ajuster la taille du conteneur si nécessaire. */
        var vratio = vw / vh;
        var eratio = ew / eh;

        if (vratio != eratio){
            video.height = ew / vratio;
        }
    }
```

En utilisant les propriétés `width` *et* `height` *dans la fonction* `adjustVideo()`*, les valeurs obtenues sont celles précisées dans les attributs éponymes. Cependant, si ces valeurs ne sont pas définies ou si elles sont indiquées en pourcentage, les dimensions retournées ne seront pas exploitables. C'est pourquoi nous préférons les propriétés* `offsetWidth` *et* `offsetHeight` *du DOM.*

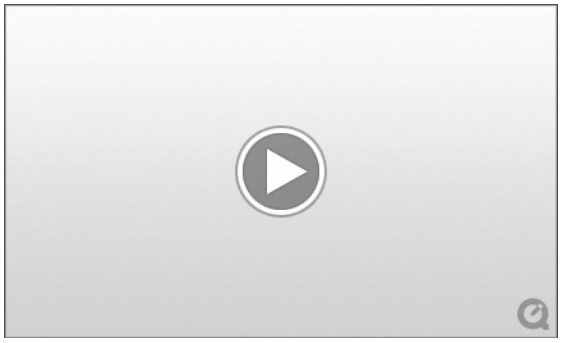
Si le navigateur n'a pas téléchargé un minimum d'informations permettant d'identifier la vidéo, les données spécifiques à cette vidéo ne sont pas disponibles. L'attribut `preload`, qui indique que les métadonnées de la vidéo doivent être préchargées avec la page HTML, n'est actuellement pas pris en charge par Safari Mobile, principalement pour limiter le trafic HTTP. Ces informations ne seront donc accessibles qu'après le déclenchement de l'événement `loadedmetadata`. L'API associée aux nouvelles balises multimédias de HTML5 propose de nombreux événements spécifiques, comme `timeupdate`, qui nous permettront de gérer les sous-titres. Nous y reviendrons plus loin dans ce chapitre.

Emplacement réservé pour la vidéo

Sur Safari Mobile, aucune donnée concernant la vidéo n'est téléchargée avant qu'elle ne soit démarrée, tout au moins aucune que nous pouvons exploiter. En attendant que ces données arrivent, l'image définie dans l'attribut `poster` remplace provisoirement la vidéo en conservant son rapport largeur/hauteur. Bien entendu, cette image doit être aussi représentative de la vidéo que possible. Si aucune image n'est indiquée ou si l'URL est invalide, l'iPhone et l'iPod touch, avec un système d'exploitation antérieur à la version 3.2, affichent un dégradé bleu à l'emplacement réservé à la vidéo (voir Figure 8.2).

À partir de la version 3.2, lorsque l'attribut `poster` n'est pas défini, l'intégralité de la zone de la vidéo apparaît en noir, sans aucune indication. Cela vient probablement du fait que l'iPad permet de visionner des films directement dans le navigateur, sans démarrer le lecteur QuickTime en plein écran comme dans les versions antérieures. Vous pouvez donc créer du contenu multimédia qui s'intègre à vos pages, le noir plus neutre simplifiant les problèmes de rendu.

Figure 8.2
L'image qui remplace par défaut la vidéo sur l'iPhone.



Lire la vidéo

Sur l'iPhone et l'iPod touch, l'utilisateur doit taper sur l'emplacement réservé pour démarrer la lecture de la vidéo. Un lecteur externe est alors ouvert (voir Figure 8.3). Puisque nous quittons le navigateur, vous pourriez supposer que tous les événements associés à la balise disparaissent. C'est faux. L'utilisation d'un lecteur externe n'empêche pas les actions ni les événements associés à la vidéo d'être exécutés, comme le redimensionnement du conteneur.

Figure 8.3
Le lecteur QuickTime sur l'iPhone.

Sur l'iPad, la balise `<video>` ne donne pas le même résultat. La vidéo n'est pas démarrée automatiquement et rien ne se produit lorsque l'utilisateur tape sur la zone réservée. Pour lancer la lecture de la vidéo, il faut soit ajouter l'attribut `controls`, qui indique au lecteur d'afficher les boutons de contrôle, soit invoquer la méthode `play()` sur l'objet vidéo. Dans la première solution, le lecteur affiche un bouton de lecture. Lorsque les contrôles sont activés, ils recouvrent une partie de la vidéo (voir Figure 8.4 page suivante), mais disparaissent automatiquement au bout de quelques secondes. À l'instar de la plupart des applications multimédias modernes, son utilisation est fluide.

```
<video src="mavideo.mp4" poster="apercu.png" id="mavideo"
    width="320" height="240" controls></video>
```

```
<!-- Gérer la lecture en JavaScript. -->
<button onclick="document.getElementById('mayvideo').play()">
    Visionner la vidéo
</button>
```

Figure 8.4
Lorsque les contrôles sont activés, un bouton de lecture apparaît sur la vidéo. Un appui sur ce bouton démarre la vidéo et affiche d'autres contrôles.

L'élément vidéo ne prend pas en charge lui-même l'événement `onclick` afin de ne pas entrer en conflit avec ces contrôles, même s'il est inclus dans un autre conteneur. C'est pourquoi il faut ajouter un bouton séparé à l'extérieur de l'emplacement réservé à la vidéo.

Quelle que soit la solution retenue, la vidéo ne sera démarrée que sur une action de l'utilisateur. L'attribut `autoplay` est pris en charge uniquement dans un cas très précis, que nous verrons plus loin dans ce chapitre. Puisque les mêmes restrictions s'appliquent à la méthode `play()`, vous ne pouvez pas utiliser `window.setTimeout()` pour lancer la lecture après un certain délai. Ce mode de fonctionnement vient probablement du fait que les réseaux cellulaires facturent encore à la quantité de données transférées. Recevoir une facture énorme pour la simple raison que le concepteur d'une application web vous a obligé à télécharger un fichier volumineux sans votre consentement ne plaide pas en faveur d'une bonne expérience utilisateur.

Incorporer du contenu audio

La balise `<embed>` était habituellement conseillée non seulement pour la vidéo mais également pour le contenu audio. Avec HTML5, cela a changé. Les développeurs disposent à présent d'une solution élégante :

```
<audio src="monaudio.mp3" id="monaudio"></audio>
```

L'interface DOM associée se nomme `HTMLAudioElement`. Il est très facile de créer dynamiquement un objet audio dans du code JavaScript, non seulement en invoquant la méthode du DOM `createElement()`, mais également en employant un nouveau constructeur HTML5 qui prend en argument une URL source. Grâce à cette méthode, puisque seul le son est impliqué, l'élément peut être initialisé et joué immédiatement sans être ajouté à l'arborescence du document.

```
var audio1 = document.createElement("audio");    // Façon DOM.
var audio2 = new Audio("monaudio.mp3");          // Façon HTML5.
```

Puisque aucune image ni séquence ne doit être affichée, les attributs `width` et `height` ne sont pas pris en charge. Le bloc par défaut fait 200 × 16 pixels. Sur l'iPhone et l'iPod touch, le même élément de représentation que celui de la balise `<video>` est affiché dans ces dimensions. Pour démarrer la lecture du contenu audio, il suffit de taper dessus. Ce conteneur peut également être redimensionné à l'aide des styles. Il s'agit de la seule façon d'écouter un document audio sur l'appareil sans passer par JavaScript.

Sur l'iPad, la configuration est un tantinet différente. Par défaut, la balise `<audio>` s'affiche sous forme d'une zone vide, qui sera remplacée par les contrôles appropriés en ajoutant l'attribut `controls` (voir Figure 8.5). Lorsque les contrôles sont activés, le conteneur est affiché dans les dimensions par défaut ou dans celles indiquées par des règles de style. Attention, toutefois, car la largeur des contrôles peut être augmentée, non leur hauteur ! Si vous précisez une hauteur supérieure à 16 pixels, les contrôles sont simplement alignés verticalement dans l'emplacement défini.

Figure 8.5
Les contrôles audio sur l'iPad

Dans tous les cas, le lecteur audio n'est pas particulièrement attrayant. En trouvant une solution alternative, vous pourriez améliorer l'expérience utilisateur et l'homogénéiser sur les divers appareils – sans avoir recours à la propriété `DOMWindow navigator.platform` ou `navigator.userAgent`. Il suffit d'associer un événement à une image, comme la couverture de l'album, et d'appeler la méthode `play()` sur la balise `<audio>`.

Rester raisonnable

Vous l'avez constaté, il est à présent très facile d'incorporer du contenu audio dans les pages. Faites attention toutefois à ne pas en faire trop. Par exemple, si vous souhaitez afficher une liste des vidéos, à la manière de YouTube ou de Vimeo (voir Figure 8.6 page suivante), nous vous déconseillons d'employer une balise `<video>` pour chaque élément. Puisque, sur l'iPhone et l'iPod touch, appuyer sur le lecteur démarre QuickTime, vous pourriez être tenté de proposer cette méthode pour faciliter l'accès au contenu. Sachez toutefois que multiplier les balises de média sur la même page peut ralentir énormément la navigation et rendre le navigateur moins réactif.

Vous pouvez à la place afficher une liste d'images, chacune renvoyant l'internaute vers une page de lecture dédiée. Cela permet d'améliorer l'expérience utilisateur, notamment sur l'iPad, où la lecture de la vidéo se fait directement dans le navigateur et où l'on s'attend à un visionnage de qualité supérieure à celle offerte par une liste de vignettes.

Figure 8.6
Une liste de vidéos dans l'application web YouTube.

Prendre le contrôle sur le contenu

Nous l'avons indiqué précédemment, après que l'utilisateur a ouvert le contenu vidéo ou audio, il est possible d'obtenir des informations sur ce contenu. L'API associée aux nouvelles balises de média HTML5 propose tout un ensemble de nouveaux événements DOM qui nous permettent de suivre précisément le déroulement de la lecture du contenu multimédia.

Pour illustrer quelques possibilités offertes par cette API, nous allons développer une page qui affichera en temps réel des informations sur l'état de la vidéo en cours de visionnage. Nous utiliserons principalement l'objet `TimeRanges` et ses propriétés relatives aux objets audio et vidéo ; `TimeRanges` est employé dans l'interface de `HTMLMediaElement`, dont dérivent les objets vidéo et audio.

Cette interface est très simple : la propriété `length` précise le nombre de plages disponibles, et les méthodes `start()` et `end()` retournent les informations, en secondes, sur la plage dont l'indice est passé en paramètre. Si l'indice est invalide, c'est-à-dire hors de l'intervalle, une exception `INDEX_SIZE_ERR` est lancée.

Comprendre et utiliser les plages

L'interface de `HTMLMediaElement` donne accès à plusieurs propriétés notamment `buffered`, qui fournit des informations sur l'état des données téléchargées pour la lecture du contenu multimédia, ainsi que `played`, qui indique des plages déjà lues. Dans Safari Mobile, sachez que l'on considère souvent que la propriété `played` indique une seule plage.

La propriété `seekable`, qui se fonde également sur un objet `TimeRanges`, précise les plages que le navigateur peut parcourir. Nous ne l'utiliserons pas dans notre exemple, car elle retourne généralement une plage égale à celle obtenue à partir des propriétés `startTime` (pour l'objet vidéo) et `duration`. Ce n'est pas le cas lorsque, entre autres, le serveur web ne prend pas en charge l'en-tête HTTP `Range`, qui autorise le téléchargement partiel ; il est employé, par exemple, pour permettre aux utilisateurs de reprendre un téléchargement interrompu. Par ailleurs, dans les anciennes versions d'iOS, la plage devait correspondre au tampon disponible et une exception `INDEX_SIZE_ERR` était lancée si la valeur de la propriété `currentTime` sortait de cette plage.

Bien que sur l'iPad le contenu multimédia soit joué directement dans le navigateur, les informations sont également actualisées sur les autres appareils iOS. Néanmoins, dans les versions antérieures à la 3.2, la propriété `played` *ne contient aucune plage, ce qui ne devrait pas constituer un problème car il est généralement préférable d'employer les propriétés* `startTime` *et* `currentTime` *pour obtenir la position de lecture.*

Commençons par le balisage HTML de notre application d'exemple :

```
<!DOCTYPE html>
<html>
<head>
    <title>Démo multimédia</title>
    <meta name="viewport" content="initial-scale=1.0;
        maximum-scale=1.0; user-scalable=no">
    <link rel="stylesheet" href="styles/video.css">
    <script src="scripts/video.js"></script>
</head>

<body onload="setup()">
    <div class="info">
        <h1>Big Buck Bunny</h1>
        <p>Un film fantastique, sous licence Creative Commons, disponible
           en téléchargement gratuit.</p>
    </div>

    <div class="video">
        <div class="media">
            <time>Merci de patienter...</time>
            <video src="media/BigBuckBunny_640x360.m4v"
                width="100%" height="360" controls></video>
        </div>
        <div id="buffered"><meter></meter><mark></mark></div>
        <div id="played"></div>
    </div>
</body>
</html>
```

Notez que le d'identifiant buffered a deux enfants. Le premier sera une barre de progression associée au tampon, tandis que le second contiendra une flèche indiquant la position de lecture courante. Les plages lues seront ajoutées au d'identifiant played.

Nous créons ensuite une feuille de style pour les pages de vidéos, que nous nommons sans surprise video.css :

```
body {
    background-color: #444;
    color: white;
    font-family: helvetica;
    text-shadow: black 2px 2px 2px;
}

.video { margin-right: 200px; }
.media { position: relative; }

.media time {
    position: absolute;
    top: 0;
    left: 0;
    right: 0;
    background-color: rgba(0, 0, 0, 0.5);
    padding: 5px;
    color: white;
    font: bold 10px verdana;
}

.media video {
    display: block;
    background: black;
    -webkit-box-shadow: 0 4px 20px black;
}

.info {
    float: right;
    width: 190px;
    font-size: 13px;
}

.info h1 { font-size: 18px; }

#buffered, #played {
    position: relative;
    background-color: black;
    -webkit-box-shadow: 0 4px 20px black;
    margin: 1px;
}
```

```
#buffered {
    background-color: #444;
}

/* Barres de progression. */
meter, mark {
   display: block;
   height: 10px;
}

#buffered meter {
   background-color: gray;
  -webkit-box-sizing: border-box;
   border:1px solid black;
   border-width:0 1px;
}

/* Indicateur en forme de flèche. */
#buffered mark {
   position: absolute;
   border: 5px solid transparent;
   border-bottom-color: white;
   margin: -10px 0 0 -5px;
   height: 0;
}
```

Enfin, tout cela serait inutile sans le code d'affichage des informations attendues. Nous créons le fichier `video.js` dans lequel nous ajoutons la fonction `adjustVideo()` décrite précédemment. Puis nous écrivons une première fonction appelée suite à l'événement `onload` sur le document. Elle initialise les observateurs.

```
function setup() {
    var video = document.getElementsByTagName("video")[0];

    window.setInterval(checkBuffered, 500, video);
    video.addEventListener("timeupdate", checkPlayed, false);
    video.addEventListener("loadedmetadata", adjustVideo, false);
}
```

Chaque 500 millisecondes, la fonction `checkBuffered()` examine l'état du tampon et l'affiche sous forme d'une barre de progression grise sous la vidéo. La version de bureau de Safari peut profiter d'une quantité de mémoire importante, mais, sur les appareils mobiles, le navigateur est moins pourvu et le tampon change constamment sans contenir l'intégralité du film. Notez également que nous utilisons un minuteur car l'événement `progress`, qui correspond à l'actualisation du tampon, n'est pas encore pris en charge.

```
function checkBuffered(video) {
    if (video.buffered.length && video.duration) {
        var pos = calcPosition(video.buffered.start(0),
            video.buffered.end(0), video.duration);
```

```
            var bar = document.getElementById("buffered");
            updateBar(bar.firstChild, pos);
        }
    }
```

La fonction `checkPlayed()` affiche la plage lue, de nouveau à l'aide d'une barre de progression sous la vidéo. Dans Safari Mobile, puisqu'il n'y a jamais plus d'une plage déjà jouée qui va du début de la vidéo jusqu'à la position de lecture courante, une seule barre est affichée. En revanche, avec Safari, chaque plage est représentée indépendamment.

Une nouvelle plage est ajoutée lorsque, par exemple, l'utilisateur passe à une autre trame de la vidéo en utilisant le curseur. Lorsque deux plages se chevauchent, elles sont fusionnées en une seule de manière à réduire leur nombre. Nous effaçons donc le conteneur `played` lors de chaque événement `updatetime`, ce qui met à jour la barre de progression.

```
function checkPlayed(event) {
    var video = event.target;

    /* Actualiser la position du curseur. */
    var progress = document.getElementById("buffered");
    progress.lastChild.style.left =
        (video.currentTime / video.duration * 100) + "%";

    /* Actualiser l'indicateur textuel de progression. */
    updateTime(video);

    /* Effacer et ajouter les plages lues existantes. */
    var container = document.getElementById("played");
    container.innerHTML = "";

    for (var i = 0; i < video.played.length; i++) {
        var pos = calcPosition(video.played.start(i),
            video.played.end(i), video.duration);
        addRange(container, pos);
    }
}

function addRange(dest, pos) {
    var bar = document.createElement("meter");
    bar.style.backgroundColor =
        "hsl(" + (pos.from * 360 / 100 | 0) + ", 100%, 50%)";

    updateBar(bar, pos);
    dest.appendChild(bar);
}
```

Les fonctions suivantes calculent la progression et contrôlent l'affichage de la barre. La première détermine le pourcentage nécessaire à la seconde pour placer correctement les barres dans le conteneur et pour représenter graphiquement la plage lue passée en paramètre à `calcPosition()`.

```
function calcPosition(start, end, max) {
    return {
        from: (start / max * 100),
        to:   (end   / max * 100)
    }
}

function updateBar(bar, pos) {
    bar.style.width = (pos.to - pos.from) + "%";
    bar.style.marginLeft = pos.from + "%";
}
```

Les derniers traitements concernent la durée. La fonction `updateTime()` évalue le temps écoulé et le temps restant, en secondes. Elle invoque ensuite la fonction `formatTime()` pour les convertir en un format plus compréhensible.

```
function updateTime(video) {
    if (video.duration) {
        var elapsed = video.currentTime - video.startTime;
        var state = video.paused ? "Pause" : "Lecture";

        document.getElementsByTagName("time")[0].textContent =
            state + ": " +
            formatTime(elapsed) + " / " +
            formatTime(video.duration) + " (-" +
            formatTime(video.duration - elapsed) + ")";
    }
}

function formatTime(time) {
    /* Détecter les heures. */
    var cut = time < 3600 ? 4 : 1;
    var str = "";

    for (var i = 2; i >= 0; i--) {
        var step = Math.pow(60, i);
        var part = (time / step) | 0; // Alternative rapide à Math.floor().

        time -= part * step;
        str += ":" + ("0" + part).substr(-2); // Ajouter un 0 initial.
    }

    return str.substr(cut);
}
```

Ouvrez à présent la page créée dans Safari Mobile, ou dans votre navigateur de développement pour voir plusieurs plages, et démarrez la vidéo. Le résultat doit être comparable à celui de la Figure 8.7 page suivante.

Figure 8.7
La durée est affichée en partie supérieure. Les barres du tampon et de la lecture sont affichées en partie inférieure.

Dans Safari Mobile, le téléchargement de la vidéo commence uniquement suite à une action de l'utilisateur. Jusqu'à ce moment-là, aucune donnée n'est disponible, y compris la durée de la vidéo. Dès que le téléchargement a débuté, les informations sont collectées et affichées, avec une représentation précise de la progression.

Événements pris en charge

Dans l'exemple précédent, nous n'avons exploité qu'un petit nombre des événements reconnus. Le Tableau 8.1 récapitule l'intégralité des événements pris en charge, sans oublier d'en donner la signification sur iOS. L'objet `Event` passé en argument aux gestionnaires ne comprend aucune propriété particulière relative au type de l'événement, comme le temps de lecture écoulé avec l'événement `timeupdate`. Ces informations sont disponibles directement au travers de l'objet qui reçoit l'événement, comme nous l'avons montré dans l'exemple (`event.target`).

Tableau 8.1 : Liste des événements reconnus par Safari Mobile

Nom d'événement	*Description*
`abort`	Le téléchargement a été interrompu avant que le navigateur ait eu le temps de le terminer.
`emptied`	Un contenu multimédia déjà initialisé a été remis dans son état initial.
`play`	La lecture a été démarrée, par exemple après que l'utilisateur a appuyé sur un bouton.
`pause`	La lecture a été suspendue, par exemple après que l'utilisateur a appuyé sur un bouton.
`loadedmetadata`	Les informations de dimension et de durée sont disponibles.
`loadeddata`	Les données nécessaires à la lecture du contenu sont disponibles.

Tableau 8.1 : Liste des événements reconnus par Safari Mobile

Nom d'événement	*Description*
`waiting`	La lecture a été arrêtée car la trame suivante n'est pas disponible.
`playing`	La lecture a démarré.
`canplay`	Des données sont disponibles pour la lecture, sans certitude qu'elle puisse se faire sans passer par un tampon.
`canplaythrough`	Les données sont disponibles pour la lecture et le navigateur estime qu'elle peut se faire sans passer par un tampon.
`seeking`	Le navigateur doit modifier la position de lecture. Cet événement est déclenché uniquement si la recherche prend plus de temps qu'il n'est raisonnable.
`seeked`	Le navigateur a terminé le déplacement vers la nouvelle position de lecture.
`timeupdate`	La position de lecture courante a changé.
`ended`	La lecture a été arrêtée car la fin du contenu a été atteinte.
`ratechange`	La vitesse de lecture a été modifiée.
`durationchange`	La durée du contenu a changé, par exemple si la source a été modifiée.
`volumechange`	Le volume a changé. Sa modification à l'aide des contrôles du lecteur ne déclenche pas cet événement.

Certains comportements associés à ces événements peuvent surprendre. `ratechange` et `volumechange` sont pris en charge sur Safari Mobile mais sont d'une utilisation limitée. Les propriétés associées sur les objets de média n'ont aucun effet. La modification des valeurs des propriétés `playbackRate`, `defaultPlaybackRate` et `volume`, qui doit avoir pour conséquences un changement de la vitesse de lecture et du volume, déclenche l'événement mais n'affecte pas la lecture.

Pour mettre en œuvre une fonction d'avance rapide, vous devez employer la propriété `currentTime`, mais le résultat ne sera pas aussi fluide qu'en utilisant les propriétés qui affectent directement la vitesse de lecture.

La manipulation de la propriété `currentTime` génère automatiquement un événement `seeked`. Techniquement, la propriété `seeking` passe à `true` pour indiquer qu'une recherche est en cours. L'événement `seeked` est envoyé lorsque la propriété revient à la valeur `false`. Si la procédure prend plus de temps qu'attendu, et dans ce cas uniquement, un événement `seeking` est déclenché.

Le même événement est également émis lorsque le contenu multimédia arrive à la fin et que l'attribut `loop` est défini sur la balise `<audio>` ou `<video>`. Dans ce cas, l'événement `ended` n'est jamais envoyé, puisque la lecture n'est pas censée s'arrêter. Vous devrez donc déterminer manuellement si la lecture est terminée en analysant les propriétés `currentTime` et `duration` en cas d'événement `seeked`. Cela vous sera utile si vous souhaitez limiter le nombre de lectures en boucle du contenu multimédia.