

HTML5

De la page web à l'application web

2^E ÉDITION

JEAN-PIERRE **VINCENT**

*Développeur web,
animateur de braincracking.org*

JONATHAN **VERRECCHIA**

*Développeur web chez Yelp,
créateur et animateur de html5-css3.fr*

PRÉFACE DE PAUL **IRISH**

*Google Chrome & jQuery Developer Relations,
Lead Developer of Modernizr & HTML5 Boilerplate*

DUNOD

Préface

Note historique au sujet de l'article « *Cool URLs don't change* »¹ par Sir Tim Berners-Lee (inventeur du *World Wide Web*, d'HTML, d'HTTP, des URI, et président du W3C) :
« À la fin du XX^e siècle, lorsque cet article a été écrit, “cool” était une épithète d'approbation indiquant le côté jeune et branché, la qualité ou la convenance. Avec la ruée vers la conquête de notre territoire de DNS se sont imposés des choix de noms de domaine et d'URI. Ces choix s'orientaient alors parfois plus vers le côté “cool” que vers l'utilité ou la longévité. Cet article est une tentative de rediriger l'énergie derrière la recherche du “cool”. »

Alors que Sir Tim s'efforçait de convaincre le monde du caractère « cool » des URI lisibles (il était d'ailleurs totalement précurseur et influent sur le sujet), une autre chose très cool est apparue sur le Web : HTML5.

HTML5 est non seulement le nouveau terme à la mode, mais les technologies et fonctionnalités qui se cachent derrière sont également significatives et révolutionnaires. HTML5 codifie les pratiques utilisées depuis des années et apporte les fondations des applications web du futur. Leur mise en place signifie une expérience plus agréable et plus riche pour vos utilisateurs, et en général beaucoup moins de code à écrire pour vous.

Dans ce livre vous trouverez un aperçu détaillé des fonctionnalités qu'apportent HTML5 et ses spécifications associées. Dans de nombreux domaines d'HTML5, les bonnes pratiques n'ont pas encore été définies, ce qui oblige les développeurs à se balader entre les spécifications et les blogs pour trouver comment mettre tout ça en place. Ce livre ne fait pas qu'aborder les concepts, mais il recommande également les meilleures approches pour utiliser HTML5 en production.

Pour préparer le futur, il faut commencer par ce que nous pouvons implémenter aujourd'hui. Ce livre vous apprendra à utiliser ce qui marche déjà réellement dans nos navigateurs, avec une approche très pragmatique. Puisque vous devez probablement supporter de vieux navigateurs tels qu'IE7 et 8, qui possèdent des lacunes en nouvelles fonctionnalités comme canvas, la géolocalisation et les websockets, vous trouverez dans ce livre des stratégies de développement qui permettront le déploiement d'applications HTML5 vers le public le plus large.

1. <http://www.w3.org/Provider/Style/URI.html>

J'espère que ce livre vous apportera beaucoup, et que vous serez aussi enthousiastes que moi à l'égard de la nouvelle génération du Web.

Paul IRISH
Google Chrome & jQuery Developer Relations,
Lead Developer of Modernizr & HTML5 Boilerplate
Avril 2011.

Table des matières

Préface	III
---------------	-----

Avant-propos	XIII
--------------------	------

Première partie – L'évolution de HTML4

Chapitre 1 – Introduction à HTML5	3
---	---

1.1 Quel HTML5 ?	3
------------------------	---

1.1.1 Un peu d'histoire	3
-------------------------------	---

1.1.2 Plusieurs spécifications !	4
--	---

1.1.3 C'est prêt quand ?	5
--------------------------------	---

1.1.4 Les grands principes de HTML5	6
---	---

1.2 Les grandes questions	7
---------------------------------	---

1.2.1 Interface native ou pas ?	7
---------------------------------------	---

1.2.2 La philosophie de l'amélioration progressive	9
--	---

Chapitre 2 – La syntaxe et les nouvelles balises	13
--	----

2.1 Le grand nettoyage de printemps	13
---	----

2.1.1 Le Doctype	13
------------------------	----

2.1.2 La balise html	14
----------------------------	----

2.1.3 Le charset	15
------------------------	----

2.1.4 Inclure les CSS et JavaScript sans type	15
---	----

2.2	La nouvelle syntaxe	16
2.2.1	Guillemets optionnels	16
2.2.2	Balises auto-fermantes	16
2.2.3	La casse (majuscules / minuscules)	16
2.2.4	Les balises <code>html</code> , <code>head</code> et <code>body</code>	17
2.2.5	XHTML, c'est bien	17
2.2.6	De jolis attributs booléens	17
2.3	Balises et attributs supprimés	18
2.4	Balises recyclées	20
2.4.1	La balise <code>b</code>	20
2.4.2	La balise <code>strong</code>	20
2.4.3	La balise <code>i</code>	21
2.4.4	La balise <code>em</code>	21
2.4.5	La balise <code>small</code>	21
2.4.6	La balise <code>cite</code>	22
2.5	Les nouvelles balises sémantiques	22
2.5.1	Les balises structurantes	22
2.5.2	L'heure et la date avec <code>time</code>	23
2.5.3	Les balises <code>figure</code> et <code>figcaption</code>	25
2.6	Le plan de page	26
2.6.1	Le plan de page à l'ancienne	26
2.6.2	Les balises sectionnantes HTML5	27
2.6.3	Donner un titre à nos sections	28
2.6.4	La balise <code>hgroup</code>	29
2.7	Exemple de passage d'XHTML à HTML5	29
2.7.1	Le « haut » du site	30
2.7.2	Le contenu principal de notre page : l'article	31
2.7.3	Le « bas » du site	32
2.7.4	Le code final du site en HTML5	32
Chapitre 3 – Les formulaires 2.0		35
3.1	Les nouveaux types d'entrée	35
3.1.1	Des formulaires un peu vieillissants	35
3.1.2	Les formulaires de recherche	36

3.1.3	Les adresses email	37
3.1.4	Les numéros de téléphone	38
3.1.5	Les URL	38
3.1.6	Le type date	39
3.1.7	Les types number et range	40
3.1.8	Les nuanciers de couleurs	40
3.1.9	Les suggestions	41
3.2	La validation des données	42
3.2.1	Une validation native et automatique	42
3.2.2	Personnaliser les champs erronés et requis	43
3.2.3	Désactiver la validation automatique	44
3.2.4	Rendre un champ obligatoire	44
3.2.5	Utiliser les expressions rationnelles pour la validation	44
3.3	Aidons nos utilisateurs	45
3.3.1	Sélectionner automatiquement un champ	45
3.3.2	Utiliser des indices de contenu d'un champ	46
3.3.3	Bonus : input fugueur	46
3.4	Compatibilité	47
3.4.1	Et si le navigateur ne supporte pas un nouveau type ?	47
3.4.2	Stratégie de déploiement	47
Chapitre 4 – Microdata		51
4.1	Sémantique et vocabulaire	51
4.1.1	De quoi parle-t-on ?	51
4.1.2	RDFa et Microformat	52
4.2	Implémentation	53
4.2.1	Les nouveaux attributs	53
4.2.2	L'API	54
4.2.3	Les outils	55
4.2.4	Aller plus loin	55
4.3	Le bénéfice	57

Chapitre 5 – ARIA t’emmène au pays de l’accessibilité	61
5.1 Le bon rôle	61
5.1.1 Zoner	61
5.1.2 Cette zone est vivante	65
5.1.3 Les widgets	66
5.2 HTML5 est-il accessible ?	67
5.2.1 Les nouveaux éléments posent problème	67
5.2.2 Le multimédia natif ne tient pas ses promesses	68
5.2.3 La hiérarchie des titres est cassée	69
5.2.4 Canvas n’est pas accessible	69
5.3 L’accessibilité naturelle de HTML5	70
5.3.1 Zone de l’écran	70
5.3.2 Légende et figure	71
5.3.3 Barre de progression	71
5.3.4 Formulaire	73
5.3.5 Autres gadgets	74

Deuxième partie – Les applications web : les API

Chapitre 6 – Audio et vidéo	77
6.1 Les éléments natifs	77
6.1.1 Le markup	78
6.1.2 Contrôle du flux	80
6.2 L’API JavaScript	82
6.2.1 Tester le support	83
6.2.2 Remplacer les contrôles natifs	84
6.3 Les choses qui fâchent : les formats	86
6.3.1 La trilogie de la guerre des codecs	86
6.3.2 Choisir son camp	88
6.3.3 Après le codec : les formats	89
6.4 Bon pour la production ?	90
6.4.1 L’implémentation navigateur	90
6.4.2 L’intégration avec la page	93

Chapitre 7 – Canvas	101
7.1 Les bases	101
7.1.1 L'API Canvas 2D	101
7.1.2 Mettre en place notre terrain de jeu	102
7.1.3 Le repère	103
7.1.4 Dessiner des formes basiques	104
7.2 Passons aux choses sérieuses	109
7.2.1 Une bonne pratique : la translation de contexte	109
7.2.2 Appliquer des ombres	110
7.2.3 La Text API	112
7.2.4 Les dégradés	112
7.3 Accéder aux pixels du canvas	114
7.3.1 Intégrer des images	115
7.3.2 Les mains dans le cambouis : le tableau de pixels	117
7.3.3 Création d'un filtre noir et blanc	118
7.3.4 Mouais	120
7.3.5 Sauvegarder l'image	120
7.4 Les animations et les jeux	120
7.4.1 La boucle d'animation	120
7.4.2 Déplacer un rectangle	121
7.4.3 Ajouter les contrôles pour diriger notre bloc	123
7.4.4 WebGL	124
7.5 Les alternatives	125
7.5.1 Pour Internet Explorer	125
7.5.2 SVG - Points forts et points faibles	126
Chapitre 8 – Envoyer des fichiers en glisser-déposer	129
8.1 Les anciens formulaires d'envoi	129
8.2 Les forces en présence	130
8.2.1 « Et avec ça ? » ou la sélection multiple	130
8.2.2 Glissez, déposez (soufflez)	132
8.2.3 L'API File	134
8.2.4 AJAX 2, le retour	136
8.2.5 Ça « progress » ?	137

8.3	Assemblage final	138
8.3.1	Savoir recevoir	138
8.3.2	Savoir écouter	139
8.3.3	Savoir donner	140
8.4	Aller plus loin	142
8.4.1	Gérer le curseur pendant le drop de fichiers	142
8.4.2	Styler le sélecteur de fichiers	144
8.5	On faisait comment avant ?	146
8.5.1	La sélection multiple et la progression avec Flash	146
8.5.2	Le drop avec des applets Java	146
Chapitre 9 – La géolocalisation et l’API Google Maps		149
9.1	Introduction	149
9.1.1	Exemple d’application : Petit Poucet part en randonnée	149
9.1.2	La compatibilité	150
9.1.3	Les services de géolocalisation	150
9.2	Récupérer les coordonnées	151
9.2.1	Tester le support	151
9.2.2	Récupérer la position	151
9.2.3	Les demandes d’autorisation	152
9.3	Gérer les erreurs	152
9.3.1	Les différents cas d’échec	152
9.3.2	Exemple complet de gestion des erreurs	153
9.4	Transformer votre application en véritable GPS	154
9.4.1	Les options	154
9.4.2	Suivre les déplacements	154
9.4.3	Mise en pratique	155
9.4.4	La vitesse, la boussole et l’altitude	156
9.5	Utiliser l’API Google Maps	156
9.5.1	Intégrer l’API Google Maps	157
9.5.2	Création de la page qui va contenir la carte	157
9.6	Afficher la carte Google Maps	158
9.6.1	Initialiser la carte	158

9.6.2	Ajouter la géolocalisation HTML5	159
9.6.3	Centrer la carte sur un marqueur	160
9.6.4	Petit point sur la précision	161
9.6.5	Il est temps de partir en expédition	162
9.7	Le tracé (précis !) du parcours	162
9.7.1	Éviter les bonds de perte du signal GPS	162
9.7.2	Savoir quels points relier par un Polyline	162
9.7.3	Tracé du Polyline	163
Chapitre 10	– Le Web déconnecté	165
10.1	Gérer la déconnexion	165
10.1.1	Le cache d'application (AppCache)	165
10.1.2	Le manifeste d'AppCache	167
10.1.3	Un peu de pratique	169
10.1.4	Des applications web de plus en plus proches des applications natives	171
10.1.5	La Cache API	172
10.2	Stocker les données sur le client	174
10.2.1	Le stockage simple : Web Storage	175
10.2.2	Aller plus loin	178
10.2.3	Le stockage avancé : indexedDB	181
Chapitre 11	– WebSockets : l'ère du temps réel	183
11.1	L'API WebSocket	183
11.1.1	Envoyer / recevoir	183
11.1.2	Gérer la connexion	184
11.1.3	Jouons maintenant	185
11.1.4	Étendre les fonctionnalités	187
11.2	Bénéfices, limites	188
11.2.1	Les performances	188
11.2.2	Connectivité	190
11.3	Côté serveur	191
11.3.1	La voie royale	192
11.3.2	La voie diplomatique	192
11.3.3	La voie du milieu	192

11.4 Aller plus loin	193
11.4.1 L'API Server Sent Event	193
11.4.2 On faisait comment avant ?	196
11.5 En production	198
11.5.1 Contraintes	198
11.5.2 Librairies	199
Conclusion	201
Quelques autres API non couvertes	201
Le futur des applications web	202
Et le CSS3 alors ?	204
Les outils pour HTML5	205
Tables de compatibilité	211

Annexe – Fiches pratiques

ARIA	213
Canvas 2D	215
File API	218
Formulaires 2.0	219
Géolocalisation	221
Microdata	223
Multimédia	224
L'offline (AppCache)	227
Server Sent Event	229
Web Storage	230
Syntaxe et nouvelles balises	232
WebSockets	234
XMLHttpRequest Level 2	235
Index	237

Avant-propos

Avertissement

Ce livre a été créé par des professionnels du Web à l'usage de professionnels, d'étudiants ou d'amateurs éclairés qui savent déjà coder un site web en (X)HTML et en JavaScript.

Il leur permettra d'améliorer leur site existant avec les fonctionnalités introduites par HTML5 en respectant les contraintes habituelles de production concernant :

- le support des navigateurs de Internet Explorer 6 à 10,
- le support de certains *smartphones*,
- l'accessibilité, le référencement,
- la maintenabilité du code et bonnes pratiques.

Loin des envolées lyriques des journalistes à propos de HTML5 ou des démos copiées/collées d'un blog à l'autre, les auteurs apportent leurs années d'expérience en matière de développement, utilisent déjà la plupart des fonctionnalités présentées en conditions réelles et discutent sans concessions des limites et des cas d'utilisation.

Cet ouvrage n'est pas une bible, mais un véritable guide pratique de mise en production pour le bénéfice de vos utilisateurs, et pour le plaisir des développeurs.

Attention : ce livre peut également contenir des traces d'humour.

Notice d'utilisation

Les chapitres sont indépendants les uns des autres, et peuvent donc se lire dans n'importe quel ordre. Ils abordent généralement les points suivants :

- la théorie avec des exemples de code ou une démonstration,
- une discussion sur les bénéfices et les limites,
- le support navigateur et la méthode de contournement.

La première partie présente surtout les modifications du HTML, parle de sémantique et d'accessibilité, pendant que la seconde partie explore quelques fonctionnalités des applications web qui peuvent enrichir les sites d'aujourd'hui.

Les démos de code sont toutes accessibles sur <http://livre-html5.com>. Elles fonctionnent pour la plupart sur un *smartphone*, au cas où vous passeriez une partie de votre vie dans les transports en commun à lire des livres que vous seul pouvez comprendre.

Au cours de la lecture de la seconde partie de ce livre vous rencontrerez de nombreux exemples de code JavaScript. Pour une meilleure lisibilité, les noms de variables ont été préfixés de la première lettre du type : « o » pour un *Object*, « i » pour un *Integer*, « s » pour une *String*, etc.

Convention de nommage JavaScript

```
var iVersion = 5,
    sType = 'HTML',
    oUpgradeTo = { sType: 'HTML', iVersion: 5};
```

Vous trouverez en fin d'ouvrage un index qui vous permettra de retrouver rapidement les endroits importants où nous parlons de certaines fonctionnalités.

Même entièrement lu ce livre ne vous servira pas que de décoration au bureau : il contient des fiches *En résumé* de chaque API traitée avec des exemples de code couvrant les cas d'usage, afin de retrouver rapidement ce nom de propriété ou cet attribut de balise qui vous échappe.

Un plan sans accroc

Au vu de l'historique agité et du flou artistique autour de HTML5, nous avons dû faire un choix des sujets à traiter qui n'était pas basé strictement sur la spécification W3C, mais plus pragmatiquement sur les nouveautés W3C - WHATWG qui sont déployables aujourd'hui en production. Chaque chapitre parle bien sûr de la théorie, l'illustre avec des exemples pratiques, parle des limites pour la plupart réellement constatées en production et propose des alternatives pour les anciens navigateurs.

Voici un aperçu de l'aventure qui vous attend :

- En introduction à HTML5 nous essaierons de comprendre le contexte historique de cette technologie, qui la fait évoluer, en quoi elle est importante pour l'avenir du Web (et le vôtre). Ce sera aussi l'occasion de parler de certaines pratiques comme l'amélioration progressive ou de débattre sur les interfaces natives.
- Le deuxième chapitre vous fera découvrir les changements radicaux (ou pas) que votre code HTML s'apprête à subir. Vous y découvrirez la nouvelle **syntaxe** ultra-allégée d'HTML5, les nouvelles **balises**, les balises recyclées ou supprimées et vous serez sensibilisé à la notion de plan de page.
- Le chapitre 3 va donner un coup de jeune à nos chers **formulaires**. Nous y verrons comment il est extrêmement simple d'améliorer un formulaire en lui ajoutant une couche de validation native ainsi que de toutes nouvelles fonctionnalités très pratiques

- Le chapitre 4 parle de **Microdata**, qui va faire du rêve du Web sémantique une réalité implémentable immédiatement.
- L'accessibilité sera l'objet du chapitre 5 et couvrira les parties les plus utiles de **ARIA**, compagnon officiel de HTML5. Nous y discuterons de l'accessibilité de HTML5 qui fait débat.
- Après les changements au niveau HTML, nous allons passer à la partie applications web avec toutes les nouvelles fonctionnalités accessibles via JavaScript sur un bon navigateur près de chez vous.
- Le chapitre 6 fera le point sur les capacités multimédias natives des navigateurs en parlant des balises **audio** et **vidéo**, de leur intégration avec le reste des technologies web et nous verrons si ces balises sont vraiment le *flash-killer* promis.
- Autre spécification qui fait rêver : **Canvas**, introduit dans le chapitre 7. Nous réaliserons de sublimes tracés de triangles, ainsi que des filtres graphiques et des animations à couper le souffle.
- Le chapitre 8 est un exemple pratique de glisser/déposer de fichiers depuis le bureau sur une page, qui permettra d'aborder plusieurs petites API comme la **File API**, **XmlHttpRequest 2**, le **Drag and Drop**, la sélection multiple et l'élément [progress](#).
- Le chapitre 9 vous transformera en *Big Brother* en vous plongeant au cœur de l'API de **géolocalisation** qui sert à connaître la position physique de vos utilisateurs. Nous utiliserons également l'API Google Maps afin de visualiser ces localisations.
- Le chapitre 10 va vous faire découvrir les joies des applications *offline* qui sont un élément essentiel des applications web mobiles. Vous apprendrez en détail comment utiliser l'**AppCache** ainsi que le **Web Storage** pour que vos utilisateurs puissent continuer à utiliser votre application même s'ils sont déconnectés.
- Le chapitre 11 aborde la communication client/serveur avec les deux nouveautés que sont **WebSocket**, qui donne accès à des performances inédites, et **Server Sent Event** qui va remplacer beaucoup de vos hacks actuels.
- Enfin nous concluons ce livre avec un petit aperçu de toutes les API qu'il vous restera à découvrir, quelques mots sur le CSS3, et comment commencer à mettre en place un projet HTML5 en production dès maintenant grâce à plusieurs outils très pratiques : HTML5shiv, Modernizr, HTML5Boilerplate et Initializr.

La deuxième édition

Le monde du Web bouge vite d'une année à l'autre, aussi nous avons fait quelques mises à jour depuis l'année dernière. Ce livre s'étant originalement concentré sur les API stables et émulables sur les anciens navigateurs, la plupart des chapitres n'ont subi qu'un petit lifting avec la mise à jour de la compatibilité, de la syntaxe HTML ou des API. D'autres comme la vidéo ou microdata ont vu leur conclusion radicalement changée d'une année sur l'autre. En tous les cas, vous avez entre les mains un ouvrage très actuel.

Remerciements

Croyez-le ou non, les livres ne sont pas écrits que par de vieux ermites solitaires ruminant dans leur chaumière. Les auteurs se voient donc forcés de remercier quelques personnes de leur entourage.

Jean-Pierre Vincent

- Mon ancien patron et presque ami Philippe-Emmanuel Dufour qui m'a laissé du temps pour écrire ce livre et qui m'a fait confiance quand je lui affirmais en 2007 que Flash pouvait être remplacé par JavaScript ou qu'Internet Explorer 6 n'était pas un problème pour HTML5. Si ce livre est plein d'expérience tirée d'un environnement de production, c'est aussi grâce à lui.
- Rodolphe Rimelé de Alsacrérations pour avoir cru que j'étais capable de pondre un livre sur ce sujet.
- Ma femme Carine pour supporter mes longues soirées de travail. Mon fils Alexis pour me rappeler qu'un biberon sera toujours plus important que toutes les API du monde. Si vous trouvez des coquilles, elles sont de lui, c'est un grand tapeur de clavier.
- Remerciez les premiers relecteurs et amis Julien Bordereau, Sébastien Hoarau et Philippe Jung, ils vous ont évité de longues heures d'ennui.
- Les participants et les organisateurs des soirées Paris JS, Web Performance, des conférences Paris Web où j'ai pu faire des interventions d'amateur et pour le niveau technique de l'audience dans des ambiances qui rappellent que nous ne sommes pas que des cerveaux reliés à des claviers (ce qui est plus facile à réaliser une bière à la main).
- Les intervenants du forum *web.developpez.com* pour avoir expliqué à un premier publié comme moi ce qu'ils attendaient d'un bouquin sur HTML5 en long en large et en travers.
- La centaine de blogueurs techniques ou évangélistes que je suis *via* RSS ou Twitter, pour la qualité de leur travail, leur passion et leur croyance en un Web meilleur.

Jonathan Verrecchia

- Mes parents pour leur soutien et leurs précieux conseils.
- Ma très patiente petite amie Dulcie, qui a toujours un peu de mal avec les `float`.
- Mon meilleur ami David pour nos rêveries de changer le monde.
- Je remercie également ma famille, mes amis de l'EPITA et de Sfeir, les membres de mon groupe de musique qui me permettent de m'évader de mon quotidien de *geek*, mes amis d'Erasmus (cześć Darek !), mes colocataires et notre chien Booba que je grattouille en ce moment-même, les créateurs de *South Park* et de *How I Met Your Mother*, et tous ceux que j'ai oubliés (vous serez dans mon livre sur HTML6, promis !).

- D'un point de vue plus professionnel, je remercie les lecteurs de mon blog html5-css3.fr, les utilisateurs de mon outil Initializr, Didier Girard, Paul Irish, Divya Manian, Mathieu Nebra pour son SiteDuZéro, David Allen pour sa méthode GTD, Tim Ferriss, Olivier Roland, Jorge Bucay, Google, Mozilla et Twitter.
- **Remerciement particulier : Sfeir** – Je remercie également très chaleureusement ma SSI, Sfeir, qui m'a accordé du temps pour la rédaction de ce livre. Sfeir est une SSI mettant en œuvre des technologies innovantes, notamment HTML5, ainsi que la gestion de projet agile. Nos projets majeurs en cours concernent le déploiement du *cloud computing*, la conception de RIA avec jQuery ou GWT et la création d'applications mobiles pour smartphones.

PREMIÈRE PARTIE

L'évolution de HTML4

HTML5, avant d'être un mot à la mode désignant tout ce qui bouge sans *plugin* sur une page, c'est bien sûr des modifications au niveau du code HTML lui-même. Les décisions sur l'évolution des syntaxes issues de HTML4.1 et XHTML ont été prises dans un souci de simplification et d'assouplissement des règles, d'adaptation aux pratiques existantes et de rétrocompatibilité.

La rétrocompatibilité étant assurée la plupart du temps, tout intégrateur web aujourd'hui devrait intégrer la nouvelle syntaxe dans sa maintenance ou ses nouveaux développements.

Nous allons donc voir en détail les modifications de syntaxe, l'ajout, la suppression et parfois la modification du sens de certaines balises, puis voir comment améliorer l'expérience utilisateur avec les nouveaux formulaires web. Enfin nous allons voir ce que HTML5 a prévu pour le Web sémantique avec Microdata, et nous ferons un détour par l'accessibilité en parlant d'ARIA et HTML5.

1

Introduction à HTML5

Objectif

Ce court chapitre va commencer par répondre à deux questions classiques lorsque l'on parle de HTML5 : « quoi ? », et « quand ? ». « Quoi », parce que le terme est devenu flou à force d'être utilisé, et il faut bien dire que ceux qui écrivent les spécifications entretiennent ce flou artistique. « Quand » parce qu'on entend tout et son contraire sur la maturité de HTML5.

Ce sera également l'occasion de philosopher sur l'utilisation des interfaces natives et de discuter de l'amélioration progressive.

Promis, on vous fait ça court, les dix prochains chapitres ne parlent que de technique !

1.1 QUEL HTML5 ?

1.1.1 Un peu d'histoire

En 2004, des représentants d'Apple, Opera et Mozilla qui étaient dans le consortium W3C se sont vus refuser de travailler à la standardisation des « applications web ». Leur idée était d'apporter suffisamment de fonctionnalités natives en JavaScript pour que les sites puissent rivaliser avec les logiciels de bureau. Critiquant sans pour autant quitter le W3C, ils ont alors créé de manière informelle le WHATWG – ou « WHAT Working Group » avec un logo en point d'interrogation, appréciez l'humour –, pour travailler entre fabricants de navigateurs aux spécifications de ces nouvelles API, en commençant par les formulaires, alors nommés Webforms 2.



Figure 1.1 — Les logos du W3C et du WHATWG.

En 2006, heureusement, a eu lieu la grande réconciliation : le W3C pense à changer de stratégie et à ne plus essayer de faire passer des spécifications disruptives comme une syntaxe XML stricte pour XForm et XHTML2. « Webforms 2 » et « Web application 1 » passent donc au W3C sous le nom de HTML5.

En 2009, XHTML2 est abandonné par le W3C qui sentait bien que le monde ne s'intéressait plus à cette spécification. En 2011, le WHATWG décide de renommer HTML5 en « HTML living standard » pour indiquer sa volonté d'être toujours itératif et de ne pas vouloir figer le langage, ce qui est le rôle du W3C.

1.1.2 Plusieurs spécifications !

La spécification HTML5 du W3C est aujourd'hui le fruit de la collaboration intelligente entre le W3C qui formalise et valide, et le WHATWG, qui implémente et propose. Pour ne rien simplifier, il existe en plus de la version W3C¹ deux spécifications gérées par le WHATWG : la « HTML »², dite « *living standard* », et la « Web Application 1.0 »³.

Il est illusoire et inutile d'essayer de suivre l'historique de chaque fonctionnalité, car chacune a sa vie propre : certaines sont présentes dans les trois spécifications, d'autres restent au WHATWG ou sont propres au W3C et d'autres encore ont été un temps dans le HTML5 du W3C avant d'avoir droit à leur propre spécification. De plus la coordination entre le W3C et le WHATWG à intervalles réguliers, induit que vous pouvez tomber sur un texte différent chez l'un et chez l'autre, selon le lien que vous suivez ou selon votre recherche Google.

Au cas où vous y verriez clair malgré tout, les pistes ont encore été brouillées début 2011 lorsque le W3C a dit publiquement que HTML5 était plus une marque qu'autre chose, pendant que le WHATWG a considéré qu'ils ne travaillaient plus sur HTML5, mais sur HTML tout court... ou presque, puisqu'ils l'appellent « *living standard* », ce qui peut se traduire pour les cyniques par « pas fiable » et pour les positivistes par « évolutif ».

Déprimé ? Il ne faut pas et voici pourquoi : ce qui nous intéresse réellement c'est la fonctionnalité et le niveau d'implémentation dans chaque navigateur. Ce livre répondra sans concession à cette question pour chaque thème abordé.

1. Spécification HTML5 du W3C : <http://dev.w3.org/html5/spec/>

2. Spécification HTML du WHATWG : <http://whatwg.org/html>

3. Spec Web apps 1.0 : <http://www.whatwg.org/specs/web-apps/current-work/complete/>



Figure 1.2 — Pour la première fois de son histoire, le W3C a doté une de ses technologies d'une identité visuelle.

Le mot HTML5 est devenu trop vague pour en avoir une définition précise, et au final tout dépend de votre interlocuteur :

- entre développeurs d'applications web, parler de la fonctionnalité suffit (WebSocket, Canvas...) car savoir si telle ou telle API est ou non incluse dans le HTML5 du W3C n'a pas d'importance ;
- si vous parlez de code HTML, alors il vaut mieux préciser que vous utilisez la syntaxe HTML5, qui de toute façon marche partout ;
- si vous parlez à quelqu'un qui n'est pas technique, lâchez-vous : « HTML5 » vaut bien « multimédia », « DHTML », « AJAX » ou « Web 2.0 » à d'autres époques.

HTML5 tel qu'il est employé aujourd'hui est surtout l'idée d'un Web omniprésent qui se répand sur tous les supports et qui remplace petit à petit les logiciels de bureau.

1.1.3 C'est prêt quand ?

2022. C'est la première date que la presse a retenue, et elle correspond au planning proposé au tout début par Ian Hickson, l'éditeur de HTML5 au WHATWG. À cette date-là, HTML5 devrait « disposer d'au moins deux implémentations par les navigateurs complètement interopérables », et la spécification sera alors validée. Ne jetez pas ce livre tout de suite, vous allez vite comprendre la réelle signification de cette date : les critères de qualité pour le stade final des spécifications W3C sont bien au-delà de ce dont vous avez besoin.

Par exemple ce n'est qu'en 2011, après 9 ans de révision que CSS2.1, pourtant largement utilisée en production, est passée au stade final du processus de validation W3C : « Recommandation ». Et le fait d'y passer n'a bien sûr strictement rien changé au quotidien des développeurs web qui doivent continuer à composer avec d'anciens navigateurs non conformes. En fait 10 ou 15 ans de maturation pour une spécification W3C, ça n'est pas exceptionnel.

Quand peut-on l'utiliser ? Comme pour CSS2.1, la réponse est « maintenant, mais pas tout ».

Si comme beaucoup vous devez encore assurer le support d'Internet Explorer 6 ou 7, sachez que nous compatissons et que nous avons retenu dans ce livre les fonctionnalités qui restent utilisables dans ces navigateurs, moyennant quelques astuces.

Il vous appartient pour chaque fonctionnalité que vous voulez ajouter à votre site de tester :

- l'état du support des navigateurs que vous ciblez,
- les techniques alternatives,
- si une version dégradée est acceptable,
- les avantages à utiliser HTML5.

Cela tombe bien, tous les éléments de réponse sont dans ce livre.

1.1.4 Les grands principes de HTML5

Ce qui fait l'ADN du HTML5 d'aujourd'hui provient de la vision très pragmatique que les acteurs du WHATWG avaient par rapport au W3C.

La rétrocompatibilité

Le W3C travaillait alors sur XHTML2 et XForms qui supposaient de passer à une syntaxe XML stricte. Or cette syntaxe posait problème sur Internet Explorer 6, très répandu à l'époque. Le WHATWG lui se refusait à introduire des notions qui pouvaient « casser » dans les anciens navigateurs.

En plus de l'abandon de cette syntaxe imposée, le `doctype`, les formulaires, la syntaxe raccourcie des balises `style` et `script`, le contenu des nouvelles balises `audio`, `video` ou `canvas` qui s'affiche en cas de non-support... tout a été orienté pour que les futurs anciens navigateurs lisent les nouvelles pages HTML5 sans défigurer la page.

Documenter l'existant

Plutôt que de se concentrer uniquement sur les ajouts, le WHATWG a commencé par valider et décrire ce que navigateurs et développeurs faisaient déjà.

On y retrouve du coup des fonctionnalités d'IE5 qui n'avaient jamais été documentées comme `contentEditable` et le *Drag & Drop*, ce qui facilitera l'arrivée de nouveaux navigateurs qui n'auront pas à essayer de deviner les algorithmes des autres navigateurs.

Pour des raisons de rétrocompatibilité mais aussi parce qu'ils ont constaté que les développeurs faisaient ce qu'ils voulaient avec leur HTML, HTML5 s'adapte donc à ses utilisateurs (les développeurs) et accepte une syntaxe vraiment très laxiste. C'est du coup aux navigateurs d'être capables d'accepter n'importe quoi, et à ce titre HTML5 inaugure un algorithme officiel de règles de *parsing* de la source HTML. Les fautes de syntaxe des développeurs devraient donc être comprises de la même manière par tous les navigateurs avec un moteur HTML5.

Passer aux applications web

HTML a été créé pour décrire des documents texte, mais ces dix dernières années le Web a clairement évolué vers quelque chose de plus complet : des sites de e-commerce qui nécessitent des formulaires et donc une interaction avec l'utilisateur aux sites de type Google Maps ou Google Docs qui font une concurrence directe à des logiciels de bureau, en passant par des jeux disponibles sur *smartphones*. Tout cela a été fait avec des technologies qui n'étaient pas faites pour ça.

L'idée de HTML5 c'est d'abord de formaliser des choses déjà très répandues, comme les formulaires scriptés (chapitre 3), le web sémantique (Microdata, chapitre 4) ou la lecture de contenu multimédia (chapitre 6). Une fois ces actions de base facilitées pour les développeurs, ils ont les mains libres pour rentrer de plain-pied dans le monde de l'applicatif en concurrençant des plugins comme Flash et les *applets* Java, tout autant que les applications de bureau.

D'où l'arrivée de Canvas (chapitre 7) pour la 2D et la manipulation de pixels, la gestion des actions de *Drag and Drop* et l'interaction avec les fichiers du disque dur (chapitre 8) ainsi qu'une foule d'API JavaScript permettant de répondre à certains marchés de niche comme la géolocalisation (chapitre 9), le passage au mobile avec la gestion de l'*offline* (chapitre 10) et l'arrivée des jeux web, domaine jusqu'ici réservé à Flash.

Les langages du Web sont en passe de devenir un langage universel, et en cela le mot HTML5 va bien plus loin que la spécification du W3C.

1.2 LES GRANDES QUESTIONS

Lors de l'implémentation, il y aura certains débats qui seront ravivés par l'arrivée de HTML5 sur votre site. Traitons-les ici, nous ferons souvent référence à ces notions dans les chapitres à venir.

1.2.1 Interface native ou pas ?

Que ce soit pour les nouveaux formulaires, les éléments de contrôle vidéo et audio, l'élément `data-list`, les barres de progression et on en oublie, vous avez à chaque fois un choix à faire : faut-il laisser l'interface par défaut du navigateur ?

Nous avons l'habitude d'écrire et donc de styliser à la main (ou de choisir parmi une infinité de librairies) beaucoup de ces fonctionnalités à l'intérieur de nos sites. Maintenant que HTML5 les formalise, chaque navigateur propose son propre style.

C'est un vieux débat que toutes les équipes web ont déjà dû trancher lorsqu'il s'agissait de décorer (ou pas) nos bons vieux formulaires. Si vous n'aviez pas décidé, il est temps de le faire. Voici ce qui penche en faveur des interfaces natives :

1. **L'ergonomie** : votre utilisateur a déjà vu cette interface ailleurs sur le Web et sait déjà utiliser votre lecteur vidéo ou votre sélecteur de date. Il est dans sa zone de confort, et si c'est bon pour lui, c'est bon pour vous.



Figure 1.3 — Barres de contrôles multimédia, selon le navigateur.

2. **Le temps de développement** : pas besoin de coder soi-même ou de comparer des bibliothèques, vous obtenez des interfaces complexes avec quelques octets de HTML, et vous pouvez vous concentrer sur votre code métier.
3. **La maintenance** : la bibliothèque que vous utilisez sera-t-elle compatible avec les futurs navigateurs ? Prenez-vous un risque lorsque vous la mettez à jour ? Pour les fonctionnalités de base, les navigateurs sont généralement plus stables que les bibliothèques.
4. **L'accessibilité** : utiliser des éléments natifs, c'est laisser au navigateur le soin de communiquer les bonnes informations aux technologies d'assistance et cela garantit généralement une bonne navigation au clavier. Pour plus de détails concernant le statut actuel de l'accessibilité de HTML5, reportez-vous au paragraphe 5.3 *L'accessibilité naturelle de HTML5 (un jour)*.
5. **L'internationalisation** : vous saviez qu'en japonais le mois de juillet 2011 s'écrit 2011年07月 ? Est-ce que ce sont les Anglais, les Américains ou bien les deux qui inversent jours et mois dans les dates numériques ? Les navigateurs le savent.
6. **Le multi-plateforme** : le navigateur sert d'interface entre votre fonctionnalité et le système d'exploitation où il est exécuté ; en lui donnant des éléments natifs à afficher, il peut donc adapter parfaitement votre fonctionnalité. La chose est flagrante sur mobile où la lecture sur une balise `video` ou bien la sélection d'une couleur va déclencher un affichage plein écran pour un meilleur confort utilisateur.

Voici par contre ce que l'on reproche aux interfaces natives :

1. **Elles sont moches** : tous les web designers de la planète ont fustigé les boutons radio, cases à cocher et autres `select` des formulaires web sur Windows XP. L'implémentation par Opéra 10 du sélecteur de date va continuer à leur donner raison : il a probablement été *désigné* par un développeur, ce qui est une forme d'art en soi.

2. **Elles dépareillent** : votre site a ses propres couleurs, sa taille de police, sa *font*, bref il a du chien que diable, une identité. Si vous annoncez à votre designer que vous allez lui jeter en plein milieu des éléments non stylisables, qui ont leur propre identité visuelle, et qui sont les mêmes que sur tous les autres sites, il est probable qu'il vous maudisse sur trois générations.
3. **Ce ne sont jamais les mêmes** : d'un navigateur à l'autre, d'un OS à l'autre et pire d'une version de navigateur à l'autre, les interfaces natives n'ont pas la même tête. Les habitués du *pixel-perfect* qui doivent respecter des maquettes Photoshop ne peuvent tout simplement pas compter dessus.

Finalement on a d'un côté un bonus en ergonomie et tous les avantages à faire confiance à un navigateur qui connaît bien son OS. De l'autre des problématiques de design et tous les désavantages à faire confiance à un navigateur qui n'a qu'un nombre limité de développeurs.

Prenons position : pour notre part, **tant que la fonctionnalité est là, l'ergonomie nous semble l'élément le plus déterminant et l'emporte sur le design**. C'est la relation de l'utilisateur avec votre produit qui détermine son succès. Or l'interface d'un site est déjà pour l'utilisateur une découverte, ce qui constitue un obstacle pour atteindre son objectif (acheter, s'informer, envoyer un mail...). Plus votre interface lui semblera familière, plus il se sentira en confiance et meilleur sera votre chiffre d'affaires. De la même manière qu'il est contre-productif de trop décorer des champs de formulaire ou de recréer une liste de choix multiples sans *select*, il nous semble essentiel de préférer utiliser les interfaces natives qu'il saura déjà utiliser pour les avoir déjà expérimentées ailleurs.

À partir du moment où vous estimez satisfaisante la fonctionnalité fournie par le navigateur, oubliez le design.

1.2.2 La philosophie de l'amélioration progressive

En anglais ça sonne toujours mieux : *progressive enhancement* et *graceful degradation*, à faire répéter au stagiaire trois fois la bouche pleine pendant la pause déjeuner. Blague à part ce sont deux concepts qui partent de l'idée folle qu'un site web n'est pas comme une page de journal ou une maquette Photoshop, c'est-à-dire identique au pixel près quel que soit l'environnement. D'abord c'est techniquement presque impossible ne serait-ce que par le rendu des polices qui varie selon le navigateur et le système. Mais surtout cela coûte cher d'assurer un visuel parfaitement identique entre des navigateurs et des systèmes qui ont dix ans d'écart (de IE6 sur XP, à Safari sur Mac, en passant par les *smartphones*). L'idée se répand donc doucement dans la communauté des web designers qu'il faut parfois laisser tomber l'idée du *pixel-perfect* (pixels qui n'intéressent d'ailleurs que les designers, les utilisateurs ne s'en rendent pas compte) et se concentrer plutôt sur un design global cohérent sans que la page n'apparaisse cassée.

Cela, c'est pour la partie visuelle, mais en HTML5 c'est plus délicat car on va surtout parler fonctionnalités : on peut choisir l'amélioration progressive pour des

raisons de réduction des coûts de développement mais il faut le faire en connaissance de cause et en suivant les bonnes pratiques de développement. En HTML5 plus que jamais, nous allons utiliser le développement par couches.

Prenons le cas où vous feriez un formulaire :

1. Vous commencez par la partie serveur qui va réceptionner les données et les valider, et éventuellement afficher des messages d'erreur.
2. Le formulaire étant en HTML4, vous n'aviez que des champs de type `text`, que vous pouvez décorer en CSS.
3. Vous rajoutez une validation par le navigateur des champs en rajoutant les nouveaux types (`email`, `url`...) et en marquant les champs obligatoires avec l'attribut `required`.

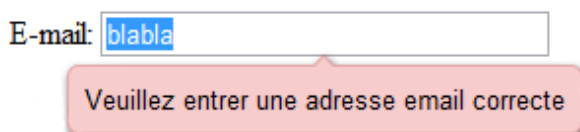


Figure 1.4 — La validation native.

Maintenant les navigateurs ne supportant pas les nouveaux formulaires auront toujours le même comportement avec des allers-retours au serveur pour valider les données, par contre vous aurez amélioré l'expérience utilisateur pour les utilisateurs des autres navigateurs, à très peu de frais.

Un autre exemple, moins visible : vous devez faire un formulaire permettant de sélectionner la marque et le modèle d'un véhicule.

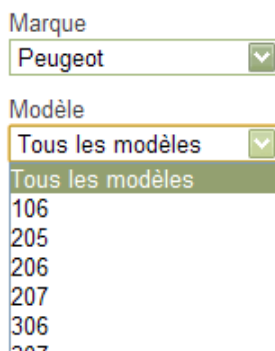


Figure 1.5 — Exemple de formulaire marque/modèle.

D'expérience les données brutes peuvent peser à elles seules 1 Mo, c'est donc le genre de données que l'on n'envoie pas d'un bloc :

1. Vous construisez donc un premier `select` contenant la liste des marques.
2. Au moment où l'utilisateur choisit une marque, vous allez chercher sur le serveur avec une XHR (requête AJAX si vous préférez) la liste des modèles de cette marque pour l'afficher dans un second `select`.
3. Si et seulement si le navigateur supporte le stockage en local, vous enregistrez cette liste chez le client.

Si l'utilisateur revient sur le formulaire en choisissant la même marque (scénario très fréquent) et que son navigateur le supporte, vous pourrez lui afficher directement les modèles de la marque, en évitant un aller / retour au serveur.

C'est cela l'amélioration progressive appliquée à HTML5 :

- Assurer pour tout le monde la fonctionnalité de base,
- Améliorer l'expérience à peu de frais pour ceux qui ont les bons navigateurs.

Enfin terminons sur un contre-exemple : l'amélioration progressive ne peut pas marcher, ou mal avec des fonctionnalités comme WebSocket. Si vous comptez rapatrier des données vitales depuis le serveur avec cette seule méthode, alors les utilisateurs des navigateurs ne supportant pas WebSocket subiront une perte d'information, ce qui n'est généralement pas tolérable.

En résumé

HTML5 se joue maintenant et de plus en plus de sites en utilisent des parties en production aujourd'hui. À l'instar de CSS 2.1 les développeurs web dépendent de l'implémentation des navigateurs et non pas de l'état des spécifications, et nous allons voir dans ce livre des parties suffisamment supportées ou simulables pour être utilisées de manière professionnelle.

2

La syntaxe et les nouvelles balises

Objectif

Lorsque quelqu'un me demande s'il peut utiliser HTML5 sur un de ses sites (attention, gros *spoiler* : la réponse est *oui* !) la question est en réalité souvent : « Est-ce que je peux utiliser les nouvelles balises HTML5 telles que `header` ou `nav` ? ».

Ces nouvelles balises sont une part essentielle de la spécification, bien qu'elles ne représentent au final qu'une très petite partie de ce qu'est HTML5. Dans ce chapitre nous allons nous intéresser à ces nouvelles balises, mais également à tout ce qui concerne la syntaxe du code HTML. Vous verrez comment HTML5 allège énormément le code et le rend plus lisible, ajoute des fonctionnalités d'accessibilité, recycle certaines balises, crée des plans de page et ajoute une bonne dose de sémantique à votre code, afin que celui-ci soit mieux compris par les machines. Rien que ça !

2.1 LE GRAND NETTOYAGE DE PRINTEMPS

2.1.1 Le Doctype

Supposons que vous commenciez un tout nouveau projet web en partant de zéro. Votre toute première action consiste à créer un fichier s'appelant probablement « `index.html` », puis à l'ouvrir dans votre IDE préféré.

Quelle est alors la deuxième action ? Écrire le Doctype du fichier ! Enfin pas exactement... mais plutôt copier/coller l'immonde syntaxe du Doctype XHTML depuis

un ancien projet pour lequel vous l'aviez d'ailleurs déjà copié/collé ! Je suis convaincu que vous avez des choses bien plus intéressantes à faire que d'apprendre par cœur ceci :

Doctype XHTML 1.0

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

Nous commençons à peine que c'est déjà compliqué ! Pour peu que vous soyez un ermite *geek* qui se crée des pages web au fin fond de la forêt sans connexion internet, vous ne pourriez même pas aller plus loin que cette étape !

Fort heureusement le W3C a pensé aux ermites *geeks* (et à tous ceux qui trouvent cette syntaxe encombrante), et a fortement simplifié le Doctype des documents HTML5. Le voici sous vos yeux ébahis :

Doctype HTML5

```
<!doctype html>
```

Eh oui, la différence est assez radicale. Ce nouveau Doctype est d'ailleurs tellement simple qu'avec un petit effort on peut même le retenir !

Compatibilité : aucun problème à signaler, y compris sur notre vieil ami Internet Explorer 6. Lui et ses successeurs ne connaissent en fait que deux modes : *quirksmode* ou mode standards, et encore ceci n'influe que sur le moteur de rendu CSS.

2.1.2 La balise html

Vous êtes prêt à continuer la création de notre nouveau projet web ? L'étape suivante est l'ouverture du premier élément : la balise `html`. Pour faire les choses correctement en XHTML, voici la ligne que nous devrions écrire :

Balise html en XHTML 1.0

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr" >
```

C'est un peu moins affreux que le Doctype vu précédemment, mais ça reste très pénible et encombrant. La seule information qui nous concerne dans cette syntaxe est la langue du document, pourquoi s'encombrer avec le reste ? L'élément racine d'un document HTML5 s'écrit simplement de cette manière :

Balise html en HTML5

```
<html lang="fr">
```

On ne garde à présent que l'information essentielle. Notre code HTML est donc plus simple et plus clair. Allez, on continue le nettoyage du code en passant au contenu de la balise `head` !

2.1.3 Le charset

Tout d'abord, il est important de spécifier l'encodage du document. Pour cela nous utilisons une balise `meta` :

Spécifier le charset en XHTML

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

Notre ermite aurait décidément la vie dure s'il devait retenir tout cela pour créer une simple page blanche ! Il n'aurait en revanche aucun mal à retenir la nouvelle syntaxe HTML5 :

Spécifier le charset en HTML5

```
<meta charset="utf-8">
```

Note : pour les plus observateurs d'entre vous qui auront remarqué l'absence du « / » de fermeture, un peu de patience, nous parlerons de la syntaxe des balises auto-fermantes un peu plus loin dans ce chapitre.

2.1.4 Inclure les CSS et JavaScript sans type

Bien ! On commence à y voir un peu plus clair. Tant que nous sommes dans le `head`, voici encore deux petits allègements bien sympathiques : il n'a en fait jamais été nécessaire de préciser le `type` d'une feuille de style CSS ni celui des scripts JavaScript puisque le CSS est le seul langage standard de styles et que le JavaScript est le seul langage standard de script.

Inclure des fichiers CSS et JavaScript en XHTML

```
<link href="style.css" rel="stylesheet" type="text/css" />
<script src="script.js" type="text/javascript"></script>
<style type="text/css"> ../.. </style>
```

Ce qui donne les lignes suivantes lorsque l'on retire le `type` :

Inclure des fichiers CSS et JavaScript en HTML5

```
<link href="style.css" rel="stylesheet" />
<script src="script.js"></script>
<style> ../.. </style>
```

Note : si vous êtes l'inventeur d'un nouveau langage de style ou de script révolutionnaire, vous pouvez sans problème rajouter l'attribut `type` car celui-ci est toujours valide, il n'est juste plus obligatoire.

2.2 LA NOUVELLE SYNTAXE

« HTML5 » n'est pas « XHTML5 », ce qui signifie que par défaut, HTML5 n'impose pas les restrictions strictes de style de code des documents XHTML. Il en résulte une syntaxe incroyablement laxiste et permissive.

2.2.1 Guillemets optionnels

Les guillemets peuvent être simples, doubles ou absents, le document reste valide HTML5. Ces syntaxes sont donc équivalentes :

```
<meta charset=utf-8>  
<meta charset="utf-8">
```

Notez cependant que les guillemets seront nécessaires si vous souhaitez spécifier plusieurs valeurs, comme c'est le cas pour l'attribut `class` :

```
<div class="classe1 classe2">
```

2.2.2 Balises auto-fermantes

Les balises auto-fermantes peuvent maintenant s'écrire sans le « / » de fermeture. Les deux exemples suivants sont donc équivalents et tout aussi valides :

```
  

```

2.2.3 La casse (majuscules / minuscules)

En HTML5 les majuscules et les minuscules sont prises en compte de la même manière. Prenons par exemple la ligne suivante écrite en minuscules :

```
<link rel="stylesheet" href="test.css">
```

Cette ligne fonctionne tout aussi bien lorsqu'elle est écrite de cette façon :

```
<Link rEl="STyLesheet" hRef="TeSt.cSS">
```

Plutôt flippant non ? Je suis sûr que certains d'entre vous ont encore des frissons de terreur lorsqu'ils voient des choses comme `<TABLE BGCOLOR="#CCCC99">` en regardant le code source d'une page HTML à l'ancienne. Bref, on *peut* utiliser des majuscules dans un document HTML5 (on peut également écrire rose sur jaune en police Comic Sans MS), mais on ne *devrait* pas le faire, pour la bonne santé de notre très cher internet.

2.2.4 Les balises `html`, `head` et `body`

Attention ! Si vous êtes un puriste de l'XHTML et que vous avez des problèmes de cœur, je vous conseille de ne pas lire ce qui suit et de passer directement à la suite. Pour les autres accrochez-vous bien avant de lire ceci :

Les balises `html`, `head` et `body` sont optionnelles.

Le document HTML5 suivant est parfaitement valide :

Document HTML5 minimal

```
<!doctype html>
<meta charset=utf-8>
<title>Salut !</title>
<p>Salut !</p>
```

Il s'affiche d'ailleurs tout à fait correctement dans tous les navigateurs, qui recréent automatiquement les balises `html`, `head` et `body` si elles sont absentes. Si cette particularité peut être assez amusante, les gens avec qui vous travaillez risquent de trouver ça plus perturbant que drôle. Il est clairement déconseillé d'utiliser ce genre de syntaxe dans un contexte autre qu'expérimental pour des raisons de maintenabilité.

2.2.5 XHTML, c'est bien

XHTML a uniformisé la syntaxe de l'HTML sur le Web grâce à des normes strictes. En HTML5 ces normes ne sont plus obligatoires, mais il est **très fortement recommandé** de continuer à les suivre afin de conserver un code propre, maintenable et facile à lire pour vous et pour les autres. L'une des forces du web est d'être *Open Source*. Chacun est libre de regarder le code source d'un site pour en apprendre un peu plus sur le standard HTML. Peut-être que vous préférez écrire vos noms de balises en majuscules et ne jamais utiliser de guillemets, mais cela pénalisera tous les curieux qui voudront « Afficher la source » de votre page si quelque chose les intrigue. Pensons à eux, respectons le standard XHTML !

2.2.6 De jolis attributs booléens

Certains attributs n'ont pas besoin de valeur particulière. Le simple fait qu'ils soient présents active leur effet. C'est le cas par exemple du nouvel attribut `autofocus` sur les champs des formulaires (expliqué dans le paragraphe 3.3.1 *Sélectionner automatiquement un champ*).

Les trois syntaxes suivantes ont le même effet :

```
<input type="text" autofocus="1" />
<input type="text" autofocus="true" />
<input type="text" autofocus="n'importe quoi en fait" />
```

Pourquoi devoir spécifier une valeur alors qu'elle ne sert à rien ? Grâce à HTML5, on peut désormais ne plus indiquer de valeur pour ces attributs booléens.

```
<input type="text" autofocus />
```

Ce qui améliore grandement la lisibilité dans le cas où il y en ait plusieurs :

```
<video autoplay loop controls>
```

Une belle petite amélioration de la syntaxe à consommer sans modération !

2.3 BALISES ET ATTRIBUTS SUPPRIMÉS

Au fil de l'évolution du standard HTML, certaines balises et certains attributs deviennent obsolètes. La modularité apportée par le CSS pousse par exemple à extraire toute information sur le rendu visuel de notre code HTML afin de n'y laisser que la donnée. Dans ce contexte, la balise `font`, qui permet d'écrire avec une police différente dans le code HTML, perd alors tout son intérêt. Elle devient donc obsolète et son utilisation est déconseillée.

Le W3C ainsi que les navigateurs ne peuvent pas se permettre de simplement dire qu'à partir de telle ou telle nouvelle version de l'HTML, cette balise ne fonctionnera plus du tout. En effet il n'est pas tolérable que sur les milliards de pages web du monde, une partie d'entre elles s'arrête tout bonnement de fonctionner correctement. Le W3C se contente donc simplement de retirer certaines balises de la nouvelle spécification, et de ne plus jamais les mentionner. Elles seront d'ailleurs déclarées invalides au test de conformité HTML5 du W3C. Les balises obsolètes continueront cependant à fonctionner car les navigateurs se forcent à assurer une rétrocompatibilité, mais vous devriez sérieusement songer à passer à de nouvelles alternatives si vous en utilisez encore. Nous les listons ici à titre de référence, nous doutons que beaucoup vous manquent.

Balises retirées de la spécification car elles ont de meilleures alternatives CSS :

- `basefont`
- `big`
- `center`
- `font`
- `strike`
- `tt`
- `u`

Balises retirées de la spécification car elles endommagent l'accessibilité :

- `frame`
- `frameset`
- `noframes`

Balises retirées de la spécification car elles sont inutilisées ou créent la confusion :

- `acronym`
- `applet`
- `isindex`
- `dir`

Attributs retirés de la spécification au profit d'autres méthodes alternatives¹ :

- `rev` et `charset` sur les balises `link` et `a`.
- `shape` et `coords` sur la balise `a`.
- `longdesc` sur les balises `img` et `iframe`.
- `target` sur la balise `link`.
- `nohref` sur la balise `area`.
- `profile` sur la balise `head`.
- `version` sur la balise `html`.
- `name` sur la balise `img`
- `scheme` sur la balise `meta`.
- `archive`, `classid`, `codebase`, `codetype`, `declare` et `standby` sur la balise `object`.
- `valuetype` et `type` sur la balise `param`.
- `axis` et `abbr` sur les balises `td` et `th`.
- `scope` sur la balise `td`.

Attributs retirés de la spécification car ils ont de meilleures alternatives CSS :

- `align` sur les balises `caption`, `iframe`, `img`, `input`, `object`, `legend`, `table`, `hr`, `div`, `h1`, `h2`, `h3`, `h4`, `h5`, `h6`, `p`, `col`, `colgroup`, `tbody`, `td`, `tfoot`, `th`, `thead` et `tr`.
- `alink`, `link`, `text` et `vlink` sur la balise `body`.
- `background` sur la balise `body`.
- `bgcolor` sur les balises `table`, `tr`, `td`, `th` et `body`.
- `border` sur les balises `table` et `object`.
- `cellpadding` et `cellspacing` sur la balise `table`.
- `char` et `charoff` sur les balises `col`, `colgroup`, `tbody`, `td`, `tfoot`, `th`, `thead` et `tr`.
- `clear` sur la balise `br`.
- `compact` sur les balises `dl`, `menu`, `ol` et `ul`.
- `frame` sur la balise `table`.
- `frameborder` sur la balise `iframe`.
- `height` sur les balises `td` et `th`.
- `hspace` et `vspace` sur les balises `img` et `object`.
- `marginheight` et `marginwidth` sur la balise `iframe`.
- `noshade` sur la balise `hr`.

1. <http://www.w3.org/TR/html5/obsolete.html#non-conforming-features>

- `nowrap` sur les balises `td` et `th`.
- `rules` sur la balise `table`.
- `scrolling` sur la balise `iframe`.
- `size` sur la balise `hr`.
- `type` sur les balises `li`, `ol` et `ul`.
- `valign` sur les balises `col`, `colgroup`, `tbody`, `td`, `tfoot`, `th`, `thead` et `tr`.
- `width` sur les balises `hr`, `table`, `td`, `th`, `col`, `colgroup` et `pre`.

2.4 BALISES RECYCLÉES

Certaines balises ont subi un remaniement. Plutôt que de les supprimer pour en introduire de nouvelles, leur fonctionnalité a été repensée, tout en évitant de perturber leur sens actuel. C'est par exemple le cas des célèbres balises `i` et `b`, qui ont fait enrager les puristes de la séparation présentation / contenu pendant de nombreuses années. Vous allez voir qu'HTML5 leur apporte une seconde vie !

2.4.1 La balise `b`

Son ancienne et terrifiante fonctionnalité était tout simplement de mettre un texte en gras. Son petit frère `strong` et son cousin `font-weight` venu tout droit de la famille CSS lui ont totalement piqué la vedette jusqu'à maintenant. Mais la balise `b` fait aujourd'hui son grand *come back* avec une toute nouvelle mission : indiquer qu'une partie de texte ressort visuellement du reste du texte. Il ne s'agit pas d'importance ou de mots-clés, mais seulement de visuel pur. Rien ne nous empêche donc de par exemple définir en CSS que le texte contenu dans une balise `b` doit s'afficher en rouge.

2.4.2 La balise `strong`

La balise `strong` a très légèrement changé de sens. Elle passe de la forte *emphase* à la forte *importance*. Concrètement, un bon exemple valant mieux qu'un long discours :

Utilisation de la balise `strong`

```
<strong>Attention</strong>, ceci est un grillage <strong>électrifié</strong>,  
vous risquez de vous prendre une <strong>grosse châtaigne</strong>.
```

`strong` est la balise la plus adaptée pour mettre en avant certains mots-clés importants de vos textes, et cela permet d'ailleurs également d'améliorer votre référencement sur ces mots-clés (un petit conseil de SEO gratuit rien que pour vous !). Selon la spécification, plus vous imbriquez de `strong`, plus son contenu est important, mais impossible de prédire si les moteurs de recherche exploiteront cette information. Quand aux lecteurs de texte, ils ne doivent pas changer de ton au moment de lire ce contenu, voyez plutôt avec la balise `em` pour cette fonctionnalité.

2.4.3 La balise i

La balise `i` s'emploie lorsque vous utilisez des mots qui ne sont pas les vôtres :

- Lorsque vous utilisez des mots étrangers ou des mots techniques : si en plein milieu d'un texte en français j'ai par exemple envie de vous dire qu'il est important de faire des *backups* de vos *datastores* avec des logiciels *user-friendly* parce que c'est plus *safe*, je me dois d'utiliser la balise `i` pour mettre ces mots en italique.
- Lorsque vous reprenez les mots de quelqu'un, ou que vous utilisez un ton différent. Par exemple si quelqu'un dit « Il faut beau aujourd'hui » et que vous souhaitez tourner sa phrase au ridicule, vous écririez le dialogue en mettant sa phrase en italique pour reprendre ses termes : « Et l'autre qui me dit *il fait beau aujourd'hui* ! Pfff... n'importe quoi ! ».

2.4.4 La balise em

La balise `em` s'emploie pour prononcer un texte avec emphase. Une même phrase toute simple peut prendre des sens très différents selon l'emphase utilisée. Nous allons reprendre un exemple du WHATWG pour illustrer ceci :

« Les chats sont mignons. »

Si nous appliquons l'emphase sur « chats » cela sous-entend que nous insistons sur le fait que le sujet principal de la discussion est l'espèce d'animal qui est mignonne. Peut-être que quelqu'un a dit dans une phrase précédente qu'il trouvait les chiens mignons :

« Les chiens sont mignons.

— Non ! Les *chats* sont mignons ! »

Les autres sens possibles en déplaçant l'emphase sont par exemple :

- « Les chats *sont* mignons » lorsque quelqu'un a dit précédemment qu'il ne les trouvait pas mignons.
- « Les chats sont *mignons* » lorsque quelqu'un a dit précédemment qu'il les trouvait paresseux.
- « *Les chats sont mignons* » lorsque vous prononcez avec insistance la phrase entière (typiquement avec beaucoup de ponctuation et de hachage).

Cette information peut être interprétée par les lecteurs d'écran.

2.4.5 La balise small

Anciennement utilisée pour indiquer qu'un texte était écrit en petit, la balise `small` a désormais une toute autre utilité. Elle sert à écrire les « petites lignes », qui sont typiquement des termes et conditions légales, ou des petites précisions en pied de page.

Exemple d'utilisation de la balise small

```
<small>* : Offre valable jusqu'à hier.</small>
```

2.4.6 La balise cite

Cette balise était utilisée pour citer un nom d'œuvre ou d'auteur. Elle ne peut désormais s'appliquer qu'aux noms d'œuvres (livres, essais, chansons, poèmes, peintures, pièces de théâtre, etc.).

Exemple d'utilisation de la balise cite

```
<p><cite>La Joconde</cite>, de Léonard de Vinci est un beau tableau.</p>
```

2.5 LES NOUVELLES BALISES SÉMANTIQUES

2.5.1 Les balises structurantes

HTML5 apporte un lot de nouvelles balises sémantiques qui nous permettent de nous affranchir de l'utilisation à outrance des balises `div` au profit de balises au nom plus explicites. Ces noms de balises n'ont pas été choisis au hasard. En 2005, Google a donné accès au W3C à l'ensemble de son index afin d'établir quels étaient les *classes* et *ids* les plus utilisés sur le Web. Cette liste a été analysée et retravaillée, et parmi les balises structurantes, voici celles qui font désormais partie de la spécification HTML5 :

- `header`
- `footer`
- `nav`
- `aside`
- `article`
- `section`

Elles constituent donc notre nouvelle trousse à outils qui vont servir à structurer notre document HTML.

Pourquoi utiliser ces nouvelles balises ?

- Elles rendent votre code plus lisible en évitant d'utiliser des *classes* et des *ids* inutiles. Pourquoi utiliser un `div` ayant pour id « header » alors que l'on peut directement utiliser une balise `header` ? Le code HTML et le CSS résultants seront plus clairs et plus facilement maintenables.
- Elles ajoutent de la sémantique compréhensible par les machines. On peut par exemple imaginer un *plugin* de navigateur qui propose de masquer le contenu des éléments de navigation ou ceux n'ayant pas de lien particulier avec le contenu. Elles permettent également de faciliter la navigation aux personnes présentant des handicaps (voir le paragraphe 5.1.1 *Zoner*).
- Elles permettent d'aider les moteurs de recherche à mieux comprendre comment est structuré votre document, et d'attribuer (probablement) un poids plus important à votre vrai contenu, plutôt qu'aux éléments non relatifs à celui-ci. Ce qui peut potentiellement améliorer votre référencement.

Interrogeons-nous à présent sur le sens de chacune d'entre elles. Les descriptions données ci-dessous sont des traductions adaptées des descriptions officielles du W3C, ce qui explique qu'elles ne soient pas très *funky* :

- **header** – Représente un ensemble pouvant inclure une introduction et/ou des éléments de navigation. Il contient généralement un titre (**h1-h6**) et peut contenir une table des matières, un formulaire de recherche, ou des logos pertinents.
- **footer** – Contient typiquement des informations sur la section à laquelle il appartient, comme par exemple qui l'a écrite, des liens vers d'autres documents ou des informations de copyright.
- **nav** – Correspond à des liens vers d'autres pages ou vers différentes parties de la page actuelle. Une balise **nav** doit contenir un bloc de navigation majeur tel qu'un menu ou un fil d'Ariane, et pas seulement une suite de liens. Par exemple, une liste de liens communs tels que l'accueil, les termes légaux et le copyright présentent en pied de page n'a pas besoin de la balise **nav**.
- **aside** – Cette balise représente une portion de page dont le contenu a une relation avec le contenu principal, sans pour autant être essentiel. Si la balise **aside** est située dans une section particulière elle traitera du même sujet que celle-ci. Si elle est au contraire en dehors d'une section, elle traitera du même sujet que le site en général. Une balise **aside** peut par exemple contenir des publicités, des citations ou des éléments de navigation.
- **article** – Un article est une portion de page qui se suffit à elle-même. C'est-à-dire qu'elle garde son sens même séparé des éléments situés autour d'elle. Elle est faite pour être indépendante et peut être par exemple lue au travers d'un lecteur de flux RSS.
- **section** – Une section est un groupement thématique de contenu incluant généralement un titre. Des exemples de sections sont par exemple les chapitres d'un livre ou des onglets. Une page d'accueil pourrait être découpée en plusieurs sections : une introduction, des news, et les informations de contact.

Note : une petite précision tout de même : à aucun moment il n'est question de placement pour ces éléments. Ils ont un sens sémantique et non visuel. C'est-à-dire qu'un **header** peut se trouver sur la gauche, un **aside** peut occuper le bas de la page, et qu'un **footer** peut s'afficher dans un *pop-up* de type *lightbox*.

2.5.2 L'heure et la date avec time

Il arrive souvent que l'on doive écrire des dates dans nos documents HTML. Pour cela quelle balise utilise-t-on ? Un `<p class="date">` ? Un `<div class="date">` ? Un `<span class="time">` ? Rien de bien satisfaisant... HTML5 nous apporte une nouvelle balise bien sympathique : la balise **time**. Celle-ci, en plus d'être plus adaptée qu'un **p** peu sémantique, permet également de faire comprendre aux machines l'heure ou la date spécifiée ! Ainsi, pour indiquer le 1er avril 2012, nous pouvons écrire le code qui suit.

Utilisation de la balise time

En `<time datetime="2012-04-01">ce jour de poisson d'avril 2012</time>`

L'attribut `datetime` permet de faire comprendre aux machines la vraie date tandis que le contenu de la balise est le texte qui sera affiché aux yeux de l'utilisateur. C'est très pratique pour afficher des dates simples pour l'humain (« posté il y a 5 minutes », mis à jour toutes les minutes) sans perte d'information pour les programmes qui lisent le code source.

On peut d'ailleurs ajouter l'attribut `pubdate` qui permet de préciser qu'il s'agit de la date de publication :

- de l'article si la balise `time` est située dans un élément `article`,
- du document entier sinon.

Utilisation de l'attribut pubdate

Article publié `<time datetime="2011-04-01" pubdate>hier</time>`.

Si l'on souhaite indiquer une heure en plus de la date, il suffit d'adapter la valeur de l'attribut `datetime` en conséquence, pour correspondre par exemple au format normalisé suivant :

[année]-[mois]-[jour]T[heure]:[minute]:[secondes]+/-[décalagehoraire]

Ce format est à écrire sans les crochets, le symbole avant le décalage horaire est un « + » ou un « - », et le décalage horaire est à indiquer en [heure]:[minute].

Spécifier une heure précise dans datetime

`<time datetime="2011-04-01T18:26:31+01:00">18h26, Paris, 1er avril 2011</time>`

Ce format, qui est d'ailleurs le même utilisé par MySQL ou PHP (constante `DATE_W3C`), n'accepte pas les dates avant l'an 0, ou les dates imprécises (2012-04 par exemple), ce qui a généré beaucoup de critiques. Si vous avez besoin d'évoquer des dates ou des périodes, vous pouvez cependant utiliser les formats suivants :

Formats de dates imprécises

La Révolution eu lieu `<time datetime="1789">en l'an de grâce 1789</time>` et on continue de célébrer `<time datetime="07-14">la fête nationale</time>` au `<time datetime="2012-07">mois de juillet</time>` en tirant des feux d'artifice `<time datetime="22:00">le soir</time>`.

Notez que même lorsque des éléments manquent, les autres se rangent toujours dans le même ordre : année, mois, jour, etc...

Dernière chose, vous pouvez indiquer des durées précises en commençant par la lettre P (comme « period »), suivi d'une ou plusieurs paires de chiffres et de lettres.

Indiquer des durées

Mettre à cuire `<time datetime="P30M">une demie heure</time>`.
Ce livre s'autodétruit dans `<time datetime="P5S">5 secondes</time>`

Les lettres peuvent symboliser les Heures (H), les minutes (M), les secondes (S) mais aussi les semaines (le W de *Week*), les jours (le D de *Day*). Il manque pour le moment les mois et années, ainsi que les dates avant l'an 0, mais il est très probable que cette partie de la spécification continue à évoluer.

2.5.3 Les balises `figure` et `figcaption`

Au cours de la lecture de ce livre, vous tomberez de nombreuses fois sur des « figures », ces petites choses qui viennent enrichir un contenu, qu'elles soient du code, des images, des schémas, etc.



Figure 2.1 — Le logo HTML5 dans toute sa splendeur.

Elles sont la plupart du temps accompagnées d'une petite légende et leur emplacement n'est pas primordial puisqu'elles ont un identifiant unique qui permet de les retrouver. La figure 2.1 est un exemple de figure, qui est placé avant le texte que vous êtes en train de lire. Elle aurait pu être située juste après, mais dans tous les cas vous l'auriez trouvée sans trop de difficulté tant qu'elle n'est pas trop éloignée du texte qui la cite.

En HTML5, nous avons la possibilité d'utiliser des figures, grâce à la balise `figure`. Tout comme dans ce livre, elle peut contenir des images, des portions de code, ou des choses plus exotiques telles que des poèmes, des extraits de texte... Toute chose venant « illustrer » le texte qui s'y réfère.

Une figure est la plupart du temps accompagnée d'une légende, qui utilise la balise `figcaption`. Il ne doit y avoir qu'une seul `figcaption` par figure, bien qu'une figure puisse contenir plusieurs images par exemple, comme l'illustre la figure 2.3.



Figure 2.2 — Les logos de la sémantique HTML5, des applications offline, et des effets 3D

Une telle *figure* serait obtenue en HTML avec le code suivant :

Une figure incluant trois images et une légende

```
<figure>
  
  
  
  <figcaption>Figure 2.3 - Les logos de la sémantique HTML5, des applications
  offline, et des effets 3D</figcaption>
</figure>
```

Note : il peut être parfois délicat de choisir entre la balise `aside`, `figure`, voire même une simple balise `img` dans le cadre d'une image par exemple. Il est important de bien comprendre les différents cas d'utilisation. Pour cela, lorsque vous avez à choisir, posez-vous simplement les questions suivantes :

- Est-ce que ce contenu est essentiel mais que sa position n'a pas d'importance ? Si oui utilisez `figure`.
- Est-ce que ce contenu n'est pas essentiel mais traite du même sujet ? Si oui utilisez `aside`.

Dans les autres cas... utilisez simplement autre chose. Si par exemple vous souhaitez placer une jolie illustration qui n'a aucun rapport avec le sujet simplement pour décorer votre texte, une balise `img` convient parfaitement.

2.6 LE PLAN DE PAGE

HTML5 porte une attention toute particulière au « plan de document » (que nous appellerons dorénavant *outline*, son petit nom en anglais). Il s'agit de la structure hiérarchique de votre page telle qu'elle est comprise par les machines. L'outline est un peu comme une table des matières de votre page. Les applications l'utilisant sont encore peu nombreuses mais il existe déjà des outils qui permettent de générer l'outline d'un document. C'est le cas de l'extension WebDeveloper pour Firefox et Chrome ou de l'outil HTML5 Outliner¹.

2.6.1 Le plan de page à l'ancienne

Avant HTML5, l'outline d'un document était généré uniquement en fonction des titres `h1` à `h6` qui établissaient approximativement la structure du plan de page. Effectuons quelques tests avec l'outil HTML5 Outliner pour voir comment cela fonctionne. Lorsque nous lui fournissons l'HTML suivant :

1. <http://gsnedders.html5.org/outliner>

Hiérarchie à l'ancienne avec h1 et h2

```
<body>
  <h1>Titre H1</h1>
  <h2>Titre H2 - 1</h2>
  <p>Paragraphe</p>
  <h2>Titre H2 - 2</h2>
</body>
```

Voici l'outline généré :

```
Titre H1
  Titre H2 - 1
  Titre H2 - 2
```

On voit que notre balise `p` n'a pas été prise en compte car elle n'est pas « sectionnante ». En revanche, la structure a été générée selon la hiérarchie de nos `h1` et `h2`.

2.6.2 Les balises sectionnantes HTML5

Nous avons à présent dans notre trousse à outils ces fameuses balises sectionnantes qui permettent d'être beaucoup plus fins dans la gestion de l'outline. Dès lors qu'elles sont ouvertes, elles créent un nouveau niveau hiérarchique dans le plan du document.

Les balises sectionnantes font partie de la spécification HTML5 et sont :

- `article`
- `section`
- `aside`
- `nav`

Cela a bien entendu du sens pour les balises `article` et `section`, ce qui est discutable pour les balises `aside` et `nav`. Enfin bon, le but n'est pas ici de critiquer les décisions du W3C (ça ne mènerait pas bien loin de toutes façons...) mais d'apprendre à se servir de tout ça. Testons donc ces balises avec HTML5 Outliner :

Les balises HTML5 sectionnantes

```
<body>
  <article>Article</article>
  <nav>Navigation</nav>
  <aside>Aside</aside>
  <section>Section</section>
</body>
```

La réponse d'HTML5 Outliner est alors :

```
Untitled Section
  Untitled Section
  Untitled Section
  Untitled Section
  Untitled Section
```

Nous observons que ces quatre balises créent bien des sections dans la hiérarchie, mais également que la balise `body` est considérée elle-même comme une section (le terme « section » ne signifiant pas ici la balise `section` mais plutôt une « portion de document »). Vous remarquerez également toutes ces sections sont détectées comme étant « Untitled » (sans titre), mais que dans le test précédent, la section `body` avait pris le nom de la balise `h1`. Intéressant !

2.6.3 Donner un titre à nos sections

En fait, chaque section devrait en théorie contenir un titre (`h1` à `h6`) qui est le titre de la section. Si cela est relativement logique et facile pour un `article` ou une `section`, ça l'est moins lorsqu'il s'agit de donner un titre à son menu de navigation, qui sera donc visible pour les utilisateurs. Considérons qu'il est convenable d'avoir une navigation sans titre et travaillons plutôt nos sections.

Lorsque l'on utilise un élément sectionnant tel que `section`, la hiérarchie des `h1` à `h6` est réinitialisée. C'est-à-dire que l'on peut (et que l'on devrait même) utiliser des `h1` pour titrer chacune des sections. En pratique un `h1` contenu dans une `section` sera compris comme étant « plus petit » qu'un `h1` situé directement dans la balise `body` par exemple.

Pour des questions de rétrocompatibilité et également en raison de l'affichage des navigateurs en mode « CSS désactivé », certains préfèrent cependant continuer à titrer leurs sections avec des `h2` ou `h3` par exemple.

Effectuons donc un nouveau test avec l'HTML suivant :

Supprimons les « Untitled sections »

```
<body>
  <h1>Titre H1 appliqué à body</h1>
  <section>
    <h1>Titre H1 de la section 1</h1>
  </section>
  <section>
    <h2>Titre H2 de la section 2</h1>
  </section>
</body>
```

La sortie d'HTML5 Outliner est la suivante :

```
Titre H1 appliqué à body
  Titre H1 de la section 1
  Titre H2 de la section 2
```

Bingo ! On voit ici bien qu'utiliser un `h1` ou un `h2` en titre de section fonctionne tout aussi bien. Lequel doit-on alors utiliser ? À terme il faudrait utiliser le `h1` pour tout titre de section selon la recommandation du W3C. C'est donc le choix le plus adapté aux puristes et à ceux qui souhaitent expérimenter l'HTML5 dans toute sa splendeur ! Ceux qui restent un peu craintifs se contenteront de conserver la hiérarchie de leurs titres actuelle, en l'appliquant dans des balises `section` comme nous venons de le faire.

2.6.4 La balise hgroup

Enfin, il peut tout à fait arriver que vous ayez à écrire un titre suivi d'un sous-titre (le nom de votre site puis votre slogan par exemple). Dans ce cas nous utilisons un `h1` suivi d'un `h2` (ou bien un `h2` suivi d'un `h3` si un `h1` est déjà présent dans la page). Dans ce cas il y a un petit problème concernant notre outline ! Le sous-titre `h2` est vu comme une nouvelle section du document, alors que nous voudrions simplement qu'il complète le titre principal. Dans ce cas particulier, il existe une balise spécifique : `hgroup`.

`hgroup` ne doit contenir que des titres (`h1` à `h6`) et sert à faire comprendre que seul le plus gros titre doit servir à titrer la section actuelle. On l'utilise simplement de la manière suivante :

Gérer les sous-titres avec hgroup

```
<hgroup>
  <h1>Titre de la section pris en compte dans l'outline</h1>
  <h2>Sous-titre non pris en compte</h2>
</hgroup>
```

Problème résolu ! Plutôt pratique cette balise `hgroup`... Sachez cependant qu'elle est un peu controversée et qu'elle est menacée d'être retirée de la spécification. Affaire à suivre de près donc !

L'exemple de migration de code XHTML1.0 vers HTML5 de la partie suivante va vous montrer un cas typique et concret d'utilisation des sections pour constituer l'outline de document.

2.7 EXEMPLE DE PASSAGE D'XHTML À HTML5

Un peu de pratique maintenant ! Nous allons partir d'une structure XHTML existante et lui passer un coup de lasure HTML5. Voici donc la bête avec laquelle nous allons jouer :

Une structure typique de site XHTML

```
<div id="header">
  <h2>Super titre de mon blog</h2>
  <h3>Un super sous-titre descriptif</h3>
</div>
<div id="menu">
  <ul>
    <li><a href="/">Accueil</a></li>
    <li><a href="/news">News</a></li>
    <li><a href="/contact">Contact</a></li>
  </ul>
</div>
<div id="pub">Visitez le site de mon copain</div>
<div id="article">
  <div id="article-header">
    <h1>Titre de l'article</h1>
```

```

    <div class="date">1er avril 2011</div>
  </div>
  <div class="section">
    <h2>Introduction</h2>
    <p>Paragraphe d'introduction</p>
  </div>
  <div class="section">
    <h2>Développement</h2>
    <p>Paragraphe de développement</p>
  </div>
  <div class="section">
    <h2>Conclusion</h2>
    <p>Paragraphe de conclusion</p>
  </div>
  <div id="article-footer">
    <ul>
      <li><a href="http://lien-utile.com/1">Lien utile 1</a></li>
      <li><a href="http://lien-utile.com/2">Lien utile 2</a></li>
    </ul>
    <p>Ecrit par moi-même, copyright Moi-Même Corp.</p>
  </div>
</div>
<div id="footer">
  Ce blog a été réalisé par moi-même, qui travaille dans la société <a
  href="http://grosseboite.com">GrosseBoite</a>.
</div>

```

Ceci est un document XHTML1.0 valide tel que l'on peut en trouver un peu partout. Nous allons l'améliorer petit à petit pour illustrer l'utilisation des nouvelles balises.

2.7.1 Le « haut » du site

Première étape, utiliser les bons éléments pour le *header* du site :

```

<div id="header">
  <h2>Super titre de mon blog</h2>
  <h3>Un super sous-titre descriptif</h3>
</div>

```

Je suppose que vous aurez deviné que nous allons utiliser une balise `header` pour remplacer le `div` :

```

<header>
  <h2>Super titre de mon blog</h2>
  <h3>Un super sous-titre descriptif</h3>
</header>

```

Nous allons également donner un petit coup de pouce aux outils de génération de plan de page en utilisant l'élément `hgroup` comme nous l'avons vu dans la partie précédente. Nous allons également jouer les puristes de l'outline en passant notre titre principal en `h1` plutôt qu'en `h2` :

```
<header>
  <hgroup>
    <h1>Super titre de mon blog</h1>
    <h2>Un super sous-titre descriptif</h2>
  </hgroup>
</header>
```

Vient ensuite le menu. La balise `nav` nous épargne l'utilisation d'un `div` et améliore l'accessibilité de notre site :

```
<nav>
  <ul>
    <li><a href="/">Accueil</a></li>
    <li><a href="/news">News</a></li>
    <li><a href="/contact">Contact</a></li>
  </ul>
</nav>
```

Enfin, notre espace publicitaire correspond typiquement à une utilisation de la balise `aside` :

```
<aside>Visitez le site de mon copain</aside>
```

2.7.2 Le contenu principal de notre page : l'article

Notre article constitue une pièce indépendante qui garde son sens même si elle est lue en dehors de notre site. Par conséquent, nous utilisons la balise `article`. De plus, il commence par un titre et une date de publication, qui sont l'en-tête de l'article. On utilise donc une nouvelle balise `header` contenant un titre `h1` ainsi qu'une date `time` :

```
<article>
  <header>
    <h1>Titre de l'article</h1>
    <time datetime="2011-04-01" pubdate>1er avril 2011</time>
  </header>
<!-- Suite du code de l'article -->
```

Le cœur de l'article est constitué de trois grandes parties : une introduction, un développement et une conclusion. C'est donc typiquement une bonne occasion d'utiliser des balises `section`. Tout comme nous l'avons vu pour le titre du site, nous pouvons remplacer les titres de section `h2` par des titres `h1`, ce qui ne cassera pas le plan de page, puisqu'une `section` est une sous-partie de l'article :

```
<!-- Header de l'article -->
<section>
  <h1>Introduction</h1>
  <p>Paragraphes d'introduction</p>
</section>
<section>
```

```

    <h1>Développement</h1>
    <p>Paragraphe de développement</p>
  </section>
  <section>
    <h1>Conclusion</h1>
    <p>Paragraphe de conclusion</p>
  </section>
<!-- Suite du code de l'article -->

```

Enfin, on utilise un `footer` pour clôturer l'article avec des informations supplémentaires telles que son auteur et les liens de type « pour en savoir plus » :

```

<!-- Sections de l'article -->
<footer>
  <ul>
    <li><a href="http://lien-utile.com/1">Lien utile 1</a></li>
    <li><a href="http://lien-utile.com/2">Lien utile 2</a></li>
  </ul>
  <p>Écrit par moi-même, copyright Moi-Même Corp.</p>
</footer>
</article>

```

2.7.3 Le « bas » du site

Pour terminer nous appliquons simplement une nouvelle balise `footer` sur les informations relatives au site (et non pas à l'article) :

```

<footer>
  Ce blog a été réalisé par moi-même, qui travaille dans la société <a
  href="http://grosseboite.com">GrosseBoite</a>.
</footer>

```

2.7.4 Le code final du site en HTML5

Toutes ces modifications nous donnent donc ce code final valide HTML5 :

Une structure typique de site HTML5

```

<header>
  <hgroup>
    <h1>Super titre de mon blog</h1>
    <h2>Un super sous-titre descriptif</h2>
  </hgroup>
</header>
<nav>
  <ul>
    <li><a href="/">Accueil</a></li>
    <li><a href="/news">News</a></li>
    <li><a href="/contact">Contact</a></li>
  </ul>
</nav>
<aside>Visitez le site de mon copain</aside>

```

```
<article>
  <header>
    <h1>Titre de l'article</h1>
    <time datetime="2009-10-22" pubdate>October 22, 2009</time>
  </header>
  <section>
    <h1>Introduction</h1>
    <p>Paragraphes d'introduction</p>
  </section>
  <section>
    <h1>Développement</h1>
    <p>Paragraphes de développement</p>
  </section>
  <section>
    <h1>Conclusion</h1>
    <p>Paragraphes de conclusion</p>
  </section>
  <footer>
    <ul>
      <li><a href="http://lien-utile.com/1">Lien utile 1</a></li>
      <li><a href="http://lien-utile.com/2">Lien utile 2</a></li>
    </ul>
    <p>Écrit par moi-même, copyright Moi-Même Corp.</p>
  </footer>
</article>
<footer>
  Ce blog a été réalisé par moi-même, qui travaille dans la société <a
  href="http://grosseboite.com">GrosseBoite</a>.
</footer>
```

Et voilà ! Nous nous sommes débarrassés de la totalité des `divs` et de leurs `ids` pour les remplacer par des balises sémantiques. Grâce à ces balises, la lisibilité du code est grandement améliorée, et nous bénéficions de tous les avantages de l'utilisation de balises HTML5 concernant la meilleure compréhension de notre code par les machines.

En résumé

C'est une véritable bouffée d'air frais pour notre code que nous offre le W3C avec HTML5. Le code perd en verbosité et gagne en sémantique, est mieux compris par les machines et mieux lu par les humains, bref, c'est clairement une très belle avancée, et on a tout à gagner à passer à la nouvelle syntaxe et utiliser les nouvelles balises. Ces dernières fonctionnent d'ailleurs correctement (c'est-à-dire sont reconnues et stylisables) sur tous les navigateurs dits « modernes » : Internet Explorer 9, Chrome, Firefox, Safari et Opera. Concernant Internet Explorer 6, 7 et 8, elles peuvent également fonctionner à condition d'utiliser un petit script du doux nom de « HTML5shiv » dont le fonctionnement est détaillé en conclusion de ce livre. Autrement dit, vous n'avez vraiment plus de raison de ne pas vous y mettre !

3

Les formulaires 2.0

Objectif

Dans les débuts du Web, les internautes ne faisaient que consulter les informations, comme nous le faisons avec un catalogue papier de produits. Une révolution majeure a été de pouvoir interagir avec les sites en leur fournissant des informations, afin d'obtenir un contenu plus dynamique et intéressant. Cette transmission d'information s'effectue grâce aux formulaires dans lesquels il est possible d'écrire du simple texte, sélectionner un élément d'une liste déroulante ou cocher des cases (j'espère ne rien vous apprendre !).

Mais ne trouvez-vous pas étrange qu'un élément aussi central que les formulaires soit resté identique depuis si longtemps ? Les nouveaux formulaires dits « 2.0 » sont là pour changer la donne !

3.1 LES NOUVEAUX TYPES D'ENTRÉE

3.1.1 Des formulaires un peu vieillissants

Un formulaire basique contenant tous les types de champs ressemble à la figure 3.1.

Il est certes possible de styliser ce formulaire, mais l'interaction avec l'utilisateur n'est jamais très poussée. HTML5 vient enrichir ces formulaires pour les rendre plus complets, intuitifs, et sécurisés. La principale nouveauté des formulaires dits « 2.0 » consiste à un ensemble de nouveaux types pour nos champs `input`. La plupart du temps, nous utilisons dans l'attribut `type` les valeurs `text` et `password`.

Text

Password

Checkbox 1 ☒

Checkbox 2 ☒

Checkbox 3 ☐

Radio 1 ☒

Radio 2 ☐

Select

Texte long

Figure 3.1 — Les types d'entrées classiques d'un formulaire HTML 4.01.

Nous pouvons maintenant entre autres utiliser :

- `search`
- `tel`
- `email`
- `url`
- `date`
- `number`
- `range`
- `color`

3.1.2 Les formulaires de recherche

Les champs de recherche sont assez différents des champs de texte classiques, d'un point de vue de leur comportement et de leur apparence.

Q- Orange

Recent Searches

Orange

Apple

Clear Recent Searches

Figure 3.2 — Un champ de recherche et ses spécificités.

Les systèmes d'exploitation et certains logiciels comme les navigateurs utilisent des champs de recherche personnalisés. Par exemple le fait de pouvoir sélectionner un moteur de recherche via une liste déroulante dans le navigateur, ou bien pouvoir

effacer le contenu du champ en cliquant sur une petite croix. Ces champs ont la plupart du temps une icône particulière comme une loupe ou le logo d'un moteur de recherche.

Ces styles et comportements sont familiers pour les utilisateurs et sont très intuitifs. Il est donc tout à fait pertinent de les utiliser dans le cadre d'un site web. En utilisant le type `search` sur un champ, l'accessibilité devrait être améliorée car les logiciels d'assistance vont proposer des raccourcis vers ce champ. Cependant d'ici là, utilisez l'attribut `role="search"` sur le formulaire parent.

Astuce : si vous voulez vraiment maîtriser complètement le style de ce champ sur Safari et Chrome sur Mac OS, il vous faut écrire cette règle CSS :

```
input[type="search"]{  
  -webkit-appearance:textfield;  
}
```

3.1.3 Les adresses email

La valeur `email` permet d'exiger que l'utilisateur entre une adresse email. Plus besoin d'appliquer d'expression rationnelle JavaScript pour vérifier la validité d'une adresse, d'autant que ces expressions sont généralement inexactes ! Ainsi la RFC 2822 définit une expression rationnelle longue de 426 caractères, et une version plus raisonnable qui fait tout de même 133 caractères. Saviez-vous que « `toi&moi=love@very.old.museum` » est une adresse valide ? Votre navigateur lui est au courant !

De plus, vos utilisateurs équipés de *smartphones* vous seront très reconnaissants puisque le fait d'utiliser un type `email` affiche un clavier virtuel adapté aux adresses email.



Figure 3.3 — Le clavier de saisie d'adresses email sur iOS.

Note : vous pouvez personnaliser l'expression rationnelle de l'email avec l'attribut `pattern` que nous verrons plus loin, ce qui vous permet d'appliquer les mêmes limitations que dans votre code serveur.

3.1.4 Les numéros de téléphone

La valeur `tel` permet de spécifier que l'utilisateur doit entrer un numéro de téléphone. Encore une fois, le clavier virtuel des *smartphones* s'adapte en conséquence.



Figure 3.4 — Le clavier de saisie de numéros de téléphone sur iOS

3.1.5 Les URL

Jamais deux sans trois, voici le clavier affiché pour écrire dans un champ texte dans lequel est spécifié le type `url` :



Figure 3.5 — Le clavier de saisie d'adresses email sur iOS.

Ici encore, rien de superflu, seulement le minimum pour saisir rapidement une URL simple.

3.1.6 Le type date

Si vous avez besoin de demander une date à un utilisateur, certaines questions se posent. La date est-elle valide syntaxiquement ? La date existe-t-elle ? Dans quel format est-elle écrite ? Le type `date` vise à simplifier la saisie de dates en proposant un sélecteur natif, disponible en plusieurs déclinaisons. Les navigateurs Opera ainsi que iOS proposent déjà un ensemble très complet (mais pas toujours du meilleur goût) de sélecteurs de dates.



Figure 3.6 — Un sélecteur de date sur Opera.

Il existe plusieurs déclinaisons du sélecteur de date, qui varie selon le type spécifié :

- `datetime` pour le jour et l'heure avec le fuseau horaire,
- `datetime-local` pour le jour et l'heure sans le fuseau horaire,
- `date` pour sélectionner uniquement la date,
- `time` pour sélectionner uniquement l'heure,
- `week` pour la semaine,
- `month` pour le mois.

Les attributs `min`, `max` et `step` peuvent être utilisés pour définir les limites à ne pas dépasser. On peut facilement imaginer les cas d'utilisation très simples et très pratiques de ce genre de sélecteur, pour des sites de réservation de voyages, ou d'organisation d'événements.

Lorsque l'utilisateur a choisi une date, la valeur envoyée au serveur est la même que dans l'attribut `value` du champ, c'est une chaîne de caractère au format spécifié

dans la RFC 3339¹ : 2012-12-12 11:03:12+02:00. En JavaScript vous pouvez accéder directement à l'objet `Date` avec l'attribut `valueAsDate`.

Note : il n'y a pas de moyen prévu pour styliser le sélecteur de date. Cependant un ergonomiste vous dirait qu'il vaut mieux un calendrier pas très joli mais que l'utilisateur reconnaîtra immédiatement, plutôt que de devoir s'adapter à une nouvelle interface, même joliment réalisée.

3.1.7 Les types `number` et `range`

Jusqu'à présent, lorsqu'un utilisateur avait à saisir un nombre dans une page web, il devait obligatoirement le faire en utilisant le clavier. Pourtant les systèmes d'exploitation intègrent nativement depuis longtemps deux alternatives à la saisie au clavier : les *sliders* et les « champs avec des petits boutons à droite pour augmenter ou réduire » (pas évident de mettre un nom là-dessus !).

Il est à présent possible d'utiliser ces types d'entrées dans une page HTML avec les types `number` et `range`.



Figure 3.7 — Un « champ avec des petits boutons à droite » et un slider HTML5

Vous pouvez spécifier quelles sont les valeurs minimales et maximales avec les attributs `min` et `max`, ainsi que le pas avec l'attribut `step` qui prennent tous les trois des valeurs numériques. Le code final d'un « champ avec des petits boutons à droite » ressemblera donc à ceci :

Champ numérique allant de 0 à 100 par pas de 5 en partant de 50

```
<input type="number"
      min="0"
      max="100"
      step="5"
      value="50">
```

3.1.8 Les nuanciers de couleurs

Il peut arriver que vous ayez à demander à un utilisateur de sélectionner une couleur de son choix. Dans ce cas le plus pratique est de lui fournir un nuancier afin qu'il visualise graphiquement l'ensemble des couleurs qu'il peut choisir. Il existe bien sûr des sélecteurs de couleur en JavaScript mais comme d'habitude, nous préférons opter

1. <http://tools.ietf.org/html/rfc3339>

pour la solution native qui évite de charger beaucoup de JavaScript et qui propose une interface déjà connue de l'utilisateur.

Voici des exemples de nuanciers qui apparaissent lorsque l'on clique sur un champ `color` (figure 3.8).

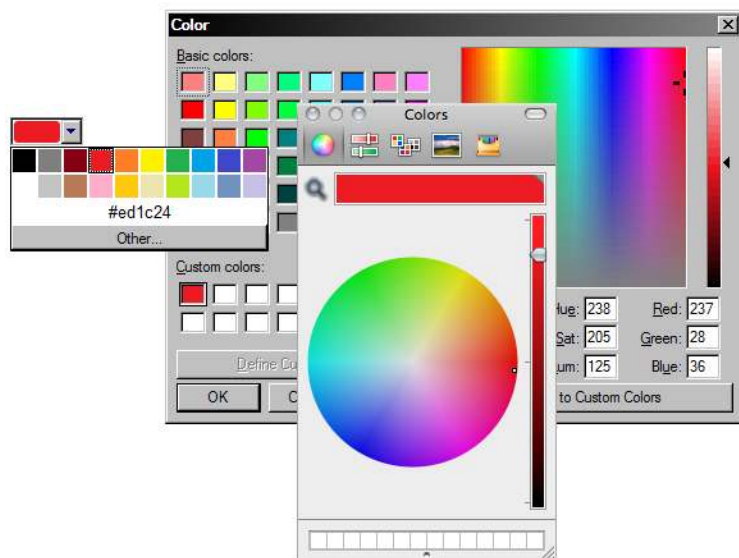


Figure 3.8 — Différents nuanciers de couleurs ouverts lorsque l'on clique sur un champ `color`.

On peut voir que ces nuanciers sont les nuanciers natifs des systèmes d'exploitation, ce qui garantit une expérience utilisateur familière. Par défaut, la valeur est `#000000`, mais une fois la couleur sélectionnée le champ contient la valeur hexadécimale de celle-ci, que l'on peut alors traiter.

3.1.9 Les suggestions

Le `datalist` est un nouvel élément qui est en quelque sorte un mélange entre une liste déroulante et un champ texte libre. Lorsque l'utilisateur commence à écrire ou double clique le champ, une liste de suggestions apparaît afin de lui proposer des entrées pertinentes.



Figure 3.9 — Les suggestions proposées par l'élément `datalist`

On bénéficie également de la précieuse auto-complétion.



Figure 3.10 — L’auto-complétion avec un `datalist`

Les différentes suggestions se déclarent avec des éléments `option`, comme pour les listes déroulantes habituelles.

Code correspondant à ce `datalist`

```
<input type="text" list="serie">
<datalist id="serie">
  <option value="Les Simpsons">
  <option value="South Park">
  <option value="Heroes">
</datalist>
```

On remarque *via* cet exemple que la `datalist` doit posséder un `id` qui est à déclarer dans le champ `input` avec l’attribut `list` correspondant. Le comportement d’une `datalist` est très familier, car il est déjà présent nativement dans les navigateurs lorsque celui-ci suggère à l’utilisateur les valeurs qu’il avait déjà entrées dans un champ. En utilisant une `datalist`, vos utilisateurs vous remercieront pour leur avoir fait gagner quelques précieuses secondes !

3.2 LA VALIDATION DES DONNÉES

3.2.1 Une validation native et automatique

Un point essentiel des formulaires 2.0 est l’apport de restrictions sur les valeurs entrées dans les champs. Si la validation finale des données doit toujours se faire côté serveur pour des raisons de sécurité, côté client vous pouvez maintenant aider l’utilisateur avec un minimum de code et une interface native. Ceci remplace les kilomètres de JavaScript que vous deviez taper pour chaque formulaire.

Si l’utilisateur entre des lettres dans un champ `number` ou `tel` par exemple, le navigateur va réagir pour le prévenir que les informations qu’il a entrées sont invalides. Il va également vérifier la validité de l’ensemble du formulaire juste avant l’envoi, afin de demander des corrections si elles sont nécessaires.

Chaque navigateur est libre de décider de l’apparence que prendront ses indicateurs. Ils seront donc par conséquent différents sur chaque navigateur.

Cette couche supplémentaire de validation nous permettra à terme de nous affranchir de la vérification des types de données en JavaScript côté client.

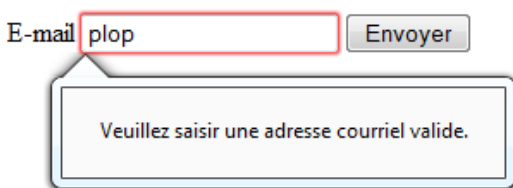


Figure 3.11 — Un indicateur d’invalidité sur Firefox

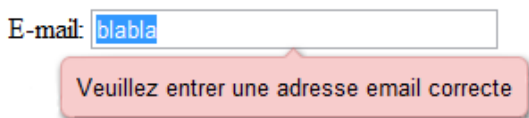


Figure 3.12 — Un indicateur d’invalidité sur Opera

3.2.2 Personnaliser les champs erronés et requis

De nouvelles pseudo-classes CSS permettent de personnaliser les champs. Par exemple, `:invalid` et `:required` permettent de les styliser lorsqu’ils sont détectés invalides ou pour indiquer qu’ils sont requis. Voici un petit exemple d’utilisation.

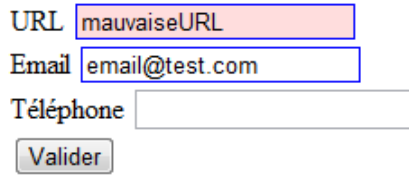
Exemple de code HTML de formulaire

```
<form>
  <label for="url">URL</label>
  <input id="url" type="url" name="url" required />
  <label for="email">Email</label>
  <input id="email" type="email" name="email" required />
  <label for="tel">Téléphone</label>
  <input type="tel" name="tel" />
  <input type="submit" />
</form>
```

Le CSS associé

```
:required{
  border:1px solid blue;
}
:invalid{
  background:#FDD;
}
```

Il existe également les pseudo-classes `:valid`, et `:optional`, qui sont sans surprise l’inverse des deux classes que nous venons de voir. `:out-of-range` et `:in-range` concernent les champs de type `date` ou `number` et enfin `:default` peut être utile pour indiquer qu’une valeur n’a pas été modifiée.



URL

Email

Téléphone

Figure 3.13 — Rendu visuel obtenu.

3.2.3 Désactiver la validation automatique

Et si l'on ne souhaite pas utiliser la vérification native du navigateur pour notre formulaire ? Rien de plus simple, l'attribut `novalidate` de la balise `form` désactive la validation native des données :

```
<form novalidate>
```

Mais qui voudrait se priver d'une telle merveille que la validation native du navigateur me direz-vous ? Eh bien si une forte contrainte d'homogénéité graphique sur tous les navigateurs vous est imposée, ou que vous trouvez les indicateurs d'erreurs des navigateurs peu esthétiques, vous devrez alors utiliser une bibliothèque JavaScript pour afficher des messages d'erreur au style identique sur tous les navigateurs. Par conséquent, afin de ne pas superposer les deux comportements, il vous faudra désactiver la validation native avec `novalidate`.

3.2.4 Rendre un champ obligatoire

En plus de spécifier le type de contenu des champs grâce aux nouveaux types, il est possible d'indiquer lesquels sont obligatoires lors de la validation du formulaire. Un champ est rendu obligatoire grâce à l'attribut `required`.

Les champs mail et pass sont rendus obligatoires

```
<input name="mail" type="email" required>  
<input name="pass" type="password" required>  
<input name="phone" type="tel">
```

Si le champ est vide au moment où l'utilisateur valide le formulaire, les messages d'erreur natifs seront affichés. Notez que les pseudo-classes ne sont pas influencées par cet attribut.

3.2.5 Utiliser les expressions rationnelles pour la validation

Supposons que vous utilisiez un champ dont le contenu n'entre dans aucune des catégories existantes, ou que vous souhaitiez définir vos propres règles concernant la validation d'un téléphone ou d'un email, afin d'être concordant avec le code serveur. Par exemple, vous faites un site sur les noms composés, et vous avez besoin de vérifier

que le champ rempli par l'utilisateur contienne bien un tiret. La solution la plus simple et la plus sûre est d'effectuer la validation de la structure de la chaîne de caractères avec une expression rationnelle (ou *regex*).

Dans notre cas nous souhaitons vérifier que chaque nom entré par l'utilisateur comporte bien un tiret mais également que chaque nom commence par une majuscule et fasse plus d'une lettre (ce qui serait un peu pénible à faire sans *regex* !). La *regex* correspondant à cette structure est :

A-Z

`[a-z]+-[A-Z][a-z]+`

Nous ne détaillerons pas le fonctionnement des expressions rationnelles, ce qui doublerait la taille de ce livre ! Contentons-nous d'utiliser cette expression, nous vous garantissons qu'elle fonctionne. L'attribut `pattern` permet d'appliquer cette *regex* au contenu de notre champ sans la moindre ligne de JavaScript :

```
<input type="text" pattern="[A-Z][a-z]+-[A-Z][a-z]+" />
```

La validation est effectuée automatiquement dès que l'utilisateur modifie le contenu du champ, ici stylisé grâce à la pseudo-classe.



Figure 3.14 — « Jonathan » ne contient pas de tiret



Figure 3.15 — « jean-pierre » ne commence pas par des majuscules



Figure 3.16 — « Jean-Pierre » est conforme à notre *regex*

3.3 AIDONS NOS UTILISATEURS

3.3.1 Sélectionner automatiquement un champ

Lorsque l'on ouvre une page HTML possédant un champ texte, il peut être pratique que le « focus » (c'est-à-dire le champ actuellement sélectionné) soit immédiatement placé sur ce champ, afin de pouvoir écrire dedans sans avoir à le sélectionner. C'est

typiquement le cas avec un moteur de recherche : l'utilisateur vient sur la page explicitement pour utiliser ce champ, donc autant lui éviter un clic inutile !

Sélectionner automatiquement un champ de recherche

```
<input type="search" autofocus>
```

Utiliser un autofocus HTML5 plutôt que l'équivalent JavaScript présente le gros avantage de ne pas provoquer de comportement non désiré auprès des utilisateurs aux besoins spécifiques, comme par exemple s'ils doivent ou souhaitent naviguer en utilisant la barre espace (oui ça arrive, ne me regardez pas comme ça !). Ces utilisateurs peuvent paramétrer leur navigateur pour accepter ou non l'autofocus, plutôt que de se voir imposer une solution JavaScript incontrôlable.

Note : dans le cas où il y ait plusieurs autofocus sur une même page, c'est le dernier dans l'ordre de la source qui a la priorité.

3.3.2 Utiliser des indices de contenu d'un champ

Le `placeholder` est le texte s'affichant dans un champ lorsque celui-ci n'est pas sélectionné et qu'il est encore vide. Jusqu'à présent il était encore une fois nécessaire d'utiliser le JavaScript pour obtenir ce comportement, et il était parfois bien difficile de le gérer dans certains cas (ceux qui ont voulu utiliser un `placeholder` dans un champ `password` savent de quoi je parle !). L'HTML5 nous simplifie grandement la vie :

Écrire « Entrez votre recherche » lorsque le champ n'est pas sélectionné

```
<input type="search" placeholder="Entrez votre recherche">
```

Note : d'après les spécifications et pour des raisons d'ergonomie et d'accessibilité, `placeholder` ne doit pas remplacer le texte du label associé au champ. Il doit rester un simple indice de ce qu'il faut entrer dans ce champ. Par exemple le label d'un champ mail doit rester « Email » (ou « courriel » voire « mél » si vous insistez), et le `placeholder` serait « nom@fournisseur.fr ».

3.3.3 Bonus : input fugueur

Je ne sais pas vous, mais personnellement je trouve que l'utilisateur est un peu trop chouchouté dans ce chapitre ! Il est temps de prendre un peu soin de nous autres développeurs : il est maintenant possible de sortir un élément de formulaire de son `form` parent. Si pour une raison quelconque vous souhaitez qu'un `input` se trouve séparé du reste de votre formulaire (pour des raisons esthétiques par exemple), vous pouvez le placer en dehors du `form`, à condition de spécifier l'`id` du `form` associé dans l'`attribut form` de l'`input`.

Exemple d'input sorti du form

```
<form id="myform">  
  <input type="submit" />  
</form>  
<input type="text" name="myinput" form="myform" />
```

Cela permet de prendre quelques libertés d'un point de vue de la structure de votre page, mais attention à ne pas trop en abuser, car non seulement cela n'est pas rétrocompatible mais votre code va aussi perdre en clarté.

3.4 COMPATIBILITÉ

3.4.1 Et si le navigateur ne supporte pas un nouveau type ?

Si l'utilisateur possède un ancien navigateur, que se passera-t-il pour tous les nouveaux types de champs ? Le navigateur en question les interprétera tout simplement comme des `input` de type `text`, c'est-à-dire ce à quoi nous sommes habitués depuis des années. Autrement dit, il n'y a absolument aucun risque à utiliser ces nouveaux types dès maintenant !

Les utilisateurs de *smartphones* vous seront reconnaissants de leur avoir facilité la tâche, et votre site sera plus ergonomique et plus accessible grâce à la validation native des données, le tout en un minimum de code.

Si vous voulez connaître précisément le support navigateur des nouvelles fonctionnalités des formulaires, il existe certains sites¹ qui maintiennent des tables de compatibilité complètes.

3.4.2 Stratégie de déploiement

Ces nouveaux formulaires sont donc parfaits si vous n'aviez pas prévu beaucoup de temps pour améliorer l'interface utilisateur : la fonctionnalité est préservée sur tous les navigateurs et l'expérience est enrichie pour les navigateurs récents et les *smartphones*. Si vous souhaitez également fournir à vos utilisateurs d'anciens navigateurs une interface enrichie sans trop vous fouler, il existe une stratégie simple qui se déroule en quatre étapes :

1. Construire votre formulaire avec la syntaxe HTML5.
2. En JavaScript, analyser le formulaire et détecter les fonctionnalités demandées.
3. Pour chaque fonctionnalité, détecter le support natif.
4. S'il n'y a pas de support, exécuter une méthode de contournement.

1. Tableau de support des formulaires HTML5 : <http://wufoo.com/html5/>

Par exemple, pour la fonctionnalité de *placeholder* :

Gestion du placeholder sur les anciens navigateurs

```
<input type="text" placeholder="Exemple" id="html5ifiez-moi">
<script>
function placeHolderification(sId) {
    // détection du support du navigateur
    var bHasSupport = ("placeholder" in document.createElement("input"));
    // ce navigateur n'a pas besoin de béquille, on arrête là
    if (bHasSupport === true)
        return;
    // à partir d'ici, on émule la fonctionnalité
    var oTheInput = document.getElementById(sId);
    // on fait disparaître la valeur lorsque l'utilisateur veut entrer quelque
chose
    oTheInput.onfocus = function() {
        this.value = "";
    };
    // la valeur réapparaît si l'utilisateur n'a finalement rien tapé
    oTheInput.onblur = function() {
        if(this.value == "")
            this.value = this.getAttribute("placeholder");
    };
}
placeHolderification("html5ifiez-moi");
</script>
```

Plusieurs remarques avant d'utiliser ce code en production :

- La détection du support des fonctionnalités peut être pénible. Ici on le fait à la main, mais si vous devez détecter la dizaine de nouvelles fonctionnalités des formulaires, vous pouvez utiliser une librairie comme Modernizr (voir conclusion de l'ouvrage).
- La fonctionnalité *placeholder* n'est pas parfaitement émulée car on ne prend pas en compte la présence d'une valeur dans le champ.
- Le texte apparaît avec une couleur par défaut, alors que les navigateurs utilisent un gris clair.
- Il n'y a pas de gestion d'erreur.

Cette fonction isolée devrait faire partie d'une classe plus grande qui analyserait tout le formulaire et les attributs pour enrichir automatiquement les formulaires en se basant uniquement sur ce que vous auriez mis dans le code HTML. Vous pouvez ainsi utiliser les sélecteurs de date et de couleur de YUI ou jQuery, récupérer le contenu des attributs `pattern` pour exécuter des expressions rationnelles, savoir quels champs sont obligatoires, etc.

Si le volume de code à taper vous fait peur, il y a deux initiatives *open source* de librairies JavaScript que vous pouvez utiliser, qui ne sont pas complètes, mais auxquelles vous pouvez contribuer s'il vous manque des fonctionnalités :

- H5F¹ qui contient l'essentiel,
- Webforms2² plus complet.

Avec ces bibliothèques ou votre adaptation, vous êtes donc paré pour utiliser les standards concernant l'écriture de formulaires (gain de temps à l'écriture du code ou lors de l'arrivée de nouvelles personnes dans l'équipe) et supporter tous les navigateurs du marché. Les navigateurs récents eux y gagneront en accessibilité et en ergonomie.

En résumé

Contrairement aux nouvelles balises structurantes que nous avons vues au chapitre précédent qui ont peu d'incidence directe sur l'utilisateur, toutes les améliorations apportées par les formulaires « 2.0 » sont immédiatement ressenties par les utilisateurs. Qu'ils soient utilisateurs de *smartphones* ou de navigateurs de bureau, ils vous remercieront d'avoir pensé à eux en rendant les formulaires plus intuitifs et plus rapides à compléter. S'il y a une nouveauté d'HTML5 qui ne coûte rien à mettre en place et qui a des bénéfices immédiats, c'est bien celle-ci !

1. Sources du projet H5F : <https://github.com/ryanseddon/H5F>

2. Page du projet Webforms2 : <http://code.google.com/p/webforms2/>

4

Microdata

Objectif

Un peu frustré par la seule introduction de la balise `article` car vous ne gérez pas un journal mais un e-commerce, un annuaire ou un répertoire de recettes ? Plutôt que de rajouter une nouvelle balise pour chaque métier, HTML5 propose aussi un standard de description d'objets et de marquage d'éléments au sein d'une page, afin que des programmes tiers puissent comprendre plus finement votre page, au premier rang desquels Google.

Cette spécification est méconnue alors qu'elle ne pose aucun problème de compatibilité et apporte à n'importe quel site des bénéfices immédiats en référencement et à terme une meilleure exploitation par des programmes tiers.

Nous allons donc voir comment introduire cette syntaxe dans vos pages et exploiter un vocabulaire déjà défini.

4.1 SÉMANTIQUE ET VOCABULAIRE

4.1.1 De quoi parle-t-on ?

Microdata, c'est un modèle de description des données avec un vocabulaire personnalisé. Ça a l'air de faire mal dit comme ça, mais cela consiste simplement à marquer ses pages d'une certaine manière pour que des robots qui comprennent le même vocabulaire sachent où trouver les informations contenues dans vos pages. Par exemple si vous avez une page décrivant un événement comme une conférence, il y a de bonnes chances pour qu'elle soit pleine de texte, de dates et d'heures. Un géant comme Google et ses milliers d'ingénieurs et scientifiques qui travaillent à l'algorithme de

recherche va peut-être réussir par recoupement à extraire les informations essentielles. Mais en marquant les données non seulement vous serez certain d'être compris par Google mais vous laissez aussi la possibilité à d'autres acteurs plus petits (moteurs et annuaires spécialisés, plug-ins et add-ons...) de comprendre quel est le nom et la date de l'événement, son site officiel, son logo, etc.

Précisons que Microdata ne définit pas de vocabulaire, mais qu'il peut réutiliser les vocabulaires déjà définis par Microformat.

4.1.2 RDFa et Microformat

L'idée de marquage et de vocabulaire n'est pas nouvelle, c'était déjà celle de RDF¹ inventé par le W3C il y a plus de 10 ans. RDF allait même beaucoup loin en permettant de décrire des relations entre objets en XML. Mais entre la difficulté d'imposer son vocabulaire aux autres, l'utilisation de XML, la complexité de la spécification, le manque d'intérêt commercial immédiat et finalement l'émergence d'API publiques pour livrer des données à d'autres, l'usage n'a pas pris sur les sites internet publics et ce sont surtout des entreprises qui l'utilisent en interne.

Les développeurs web souhaitaient tellement quelque chose de plus simple avec des vocabulaires préétablis qu'ils ont fini par créer Microformat² (μformat pour les intimes) en dehors du processus de standardisation du W3C. Le format le plus célèbre s'appelle **hCard**, et se présente sous cette forme.

Marquage d'un contact avec Microformat

```
<div class="vcard">
  <span class="fn n">Jean-pierre VINCENT</span>
  <a class="fn url" href="http://www.braincracking.org/">Braincracking</a>
  <div>Email:
    <span class="email">jp@braincracking.org</span>
  </div>
</div>
```

L'ajout des bonnes classes autour des éléments de texte dans un certain ordre permet à un lecteur de Microformat de générer une carte de visite et de repérer ici le nom complet, l'URL et le nom du blog, ainsi que l'email.

Le besoin de marquage d'élément est donc couvert par Microformat, et il est beaucoup plus simple pour les développeurs de copier / coller les exemples que de se plonger dans la spécification RDF pour savoir décrire un vocabulaire. Microformat a ainsi gagné doucement en popularité et il y a maintenant plusieurs dizaines d'objets décrits. Les robots utilisateurs de Microformat les plus connus sont les moteurs de recherche, certains mobiles, et certains logiciels de messagerie.

La contrepartie est évidente : pour marquer les données, il faut rajouter du code dans des endroits (attributs **class** et **rel**) qui n'ont pas été prévus pour cela, ce qui peut se révéler compliqué à maintenir.

1. Wiki RDF : <http://www.w3.org/RDF/>

2. Le wiki officiel de Microformat : <http://microformats.org/>

Le but de la création de Microdata est de reprendre l'idée simple d'un marquage en HTML introduit par Microformat et d'y rajouter quelques fonctionnalités tirées de RDF.

4.2 IMPLÉMENTATION

4.2.1 Les nouveaux attributs

À son niveau le plus basique, la spécification¹ n'introduit que 3 nouvelles propriétés :

1. `itemscope`, qui signale la création d'un nouvel objet, et qui est généralement une balise parente ;
2. `itemtype`, à côté de `itemscope` qui est une URL permettant de savoir quel objet est décrit ;
3. `itemprop`, qui définit une propriété de cet objet.

Prenons une page de contact d'un blogueur type, avant marquage.

Une page de contact

```
<div id="presentation">
 Salut, j'aime la vie et je m'appelle Kévina.
J'ai né il y a 15 ans et j'adore mon chat KikouLo! .
Envoyez nous <a href="mailto:kevinaDu62@skaïblog.com">un message</a> !
</div>
```

Comment une machine saurait interpréter une date de naissance là-dedans ? La personne s'appelle-t-elle Kévina ou KikouLo! ? Tête de chat ou tête d'andouille humaine ? Admettons que Kévina devienne un chef en HTML5 (ou utilise un éditeur de page préconfiguré) et rajoute un peu de Microdata à cette page pour mieux marquer les éléments.

On commence d'abord par définir une région qui représente notre objet avec `itemscope`, puis on fait référence à un vocabulaire grâce à une URL dans `itemtype`. Attention c'est subtil : pour décrire cette page de contact, nous allons utiliser le vocabulaire le plus célèbre qui est hCard tel que spécifié par le projet Microformat.

Étape 1 : marquer son territoire

```
<div id="presentation"
  itemscope
  itemtype="http://microformats.org/profile/hcard">
  ...
</div>
```

Un programme qui connaît ce vocabulaire sait donc que les enfants de cette `div` sont potentiellement des propriétés d'un objet de type hCard. Je vous invite à consulter la page <http://microformats.org/profile/hcard> pour connaître la liste des champs définis.

1. Spécification W3C de Microdata : <http://www.w3.org/TR/microdata/>

Nous allons marquer avec `itemprop` les valeurs qui nous intéressent, quitte à rajouter un `span` autour de la valeur en question.

Étape 2 : marquer les valeurs

```
<div id="presentation"
  itemscope
  itemtype="http://microformats.org/profile/hcard">
>
 Salut, j'aime la vie et je m'appelle
<span itemprop="given-name"> Kévina </span>.
J'ai né <time itemprop="bday" datetime="1996-04-08">il y a 15 ans</time> et
j'adore mon chat KikouLol .
Envoyez nous <a itemprop="email" href="mailto:kevinaDu62@skaïblog.com">un
message</a> !
</div>
```

Lorsqu'un programme repère `itemprop`, il note le nom du champ et il applique ensuite ce que le vocabulaire lui demande pour retrouver la valeur :

- Par défaut la valeur est le contenu de la balise marquée, comme ici pour `given-name`.
- Certaines valeurs comme ici `photo` et `email` peuvent être trouvées dans des propriétés de l'élément, comme ici le `src` de la balise `img`, ou le `href` de la balise `a`. C'est précisé dans les spécifications hCard.
- La récupération des dates est spécifique à Microdata, et tire parti de la nouvelle balise HTML5 `time`, très pratique pour mettre une date précise sur des mots.

Nul doute qu'avec une page aussi bien formatée, Kévina va avoir des contacts venant d'un peu n'importe qui, on n'avertit pas assez les parents sur les risques des Microdata. Un professionnel par contre aurait tout intérêt à marquer ses pages de revues, de produits, de contact et d'autres objets pour être certain d'avoir une visibilité maximale.

4.2.2 L'API

Une fois votre page enrichie avec Microdata, vous pouvez y accéder via JavaScript grâce à la méthode `document.getItems`. Du moins en théorie, car pour le moment les implémentations navigateur n'ont pas quitté les versions de développement. Vous pouvez toutefois utiliser la librairie JavaScript « Microdata.js »¹ pour émuler tout cela sur n'importe quel navigateur.

Retrouver des objets

```
// on va chercher tous les objets hCard de la page
var oList = document.getItems('http://microformats.org/profile/hcard'),
    aProperties;
for(var i=0 ; i < oList.length ; i++) {
    aProperties = oList[i].properties ;
```

1. Librairie d'émulation : <http://gitorious.net/microdatajs/microdatajs/blobs/master/microdata.js>

```
// on obtient une liste des propriétés de l'objet
for(var j = 0 ; j < aProperties.length ; j++) {
    if(aProperties[j].itemProp == 'given-name') {
        console.log(aProperties[j].itemValue ); // on a retrouvé Kévina !
    }
}
```



Voir la démo !

<http://www.livre-html5.com/microdata/getitem.php>

La méthode `getItem()` ne fait que retrouver les objets de haut niveau, il faut ensuite parcourir la propriété `properties`, qui est un tableau contenant les propriétés, dont on connaît le nom grâce à `itemProp` et la valeur grâce à `itemValue`.

C'est une manière standardisée de transmettre de l'information du *backend* au *frontend*, même si pour des données non structurées vous auriez aussi pu utiliser les attributs `data-`.

4.2.3 Les outils

Pour vérifier que votre sémantique est correcte, il existe quelques rares outils en ligne. Outre le « Google Webmaster Tools » dont nous parlons plus loin, le créateur de « microdata.js » a créé un outil en ligne¹ permettant de voir les données extraites de bout de code HTML.

Pour générer du HTML avec Microdata, il existe un autre outil en ligne hébergé par Freebase² qui prend en compte les deux vocabulaires principaux et les objets les plus utilisés (contact, événement...).

4.2.4 Aller plus loin

Associer d'autres données

Que se passe-t-il lorsque les données d'un élément sont éparpillées dans la page ? Plutôt que de se forcer à tous les mettre dans la même balise parente, vous pouvez utiliser l'attribut `itemref`. Il faut le mettre sur l'élément qui aurait dû être le parent (celui qui possède l'attribut `itemscope`) et il prend en valeur une série d'`id` des propriétés disséminées ailleurs dans la page.

Imaginons par exemple que le mail de contact soit en bas de page, et que l'on veuille associer au contact l'URL du site.

1. Tester son *markup* : <http://foolip.org/microdatajs/live/>

2. Générer un *markup* avec Microdata : <http://microdata.freebaseapps.com/index>

Associer des informations de toute la page

```
<header>
  <a href="http://braincracking.org" id="home" itemprop="url">Accueil</a>
</header>
...
<div itemscope itemtype="http://microformats.org/profile/hcard"
  itemref="email-id home">
  <span itemprop="fn">Jean-pierre VINCENT</span>
</div>
...
<footer>
  Contact: <span id="email-id" itemprop="email">jp@braincracking.org</span>
</footer>
```

Ici un parseur comprendra donc que l'objet hCard (contact) du nom de « Jean-pierre » a pour URL « braincracking.org » qui se trouvait dans l'en-tête du site, et a pour mail « jp@braincracking.org », information qui ne se trouvait que dans le pied de page.

Imbrication d'objets

L'idée vient tout droit de RDF, dont l'idée de base était de constituer naturellement une base de donnée relationnelle à la taille du Web (et de gérer la génération de graphes, mais c'est une autre histoire). Certains objets comme un contact peuvent être liés à d'autres objets, comme une organisation. Il faut donc les lier comme nous venons de le voir ou simplement les imbriquer, le tout en suivant le vocabulaire.

Prenons comme base le vocabulaire *Person* de Google.

Page contact

```
<div itemscope itemtype="http://www.data-vocabulary.org/Person/">
  Retrouvez <span itemprop="name">Jean-pierre VINCENT</span> sur le blog
  <a href="http://braincracking.org" itemprop="url">Braincracking</a>
</div>
```

Tel quel, ce HTML signifie que le contact du nom de « Jean-pierre VINCENT » a comme site personnel « braincracking.org ». Or ce n'est pas blog personnel, mais un blog professionnel où d'autres sont invités, c'est donc plus une entité morale à laquelle je suis lié. Pour signifier tout cela, on crée un deuxième objet avec *itemscope*, qui correspond à un autre vocabulaire, *Organization*.

Page contact avec objet imbriqué

```
<div itemscope itemtype="http://www.data-vocabulary.org/Person/">
  Retrouvez <span itemprop="name">Jean-pierre VINCENT</span> sur le blog
  <span itemprop="affiliation"
    itemscope itemtype="http://data-vocabulary.org/Organization/">
    <a href="http://braincracking.org" itemprop="name url">Braincracking</a>
  </span>
</div>
```

Nous avons donc rajouté un *span* pour délimiter un nouvel objet (*itemscope*) de type (*itemtype*) *Organization*, dont les deux propriétés sont contenues dans la balise

a : un nom, et une URL. Cet objet est lié (**itemprop**) à son objet parent le plus proche, qui est un objet *Person*.

Nous avons donc presque formé une phrase en langue humaine : « Je suis affilié à braincracking.org ».

4.3 LE BÉNÉFICE

Le potentiel est énorme, mais aujourd'hui il n'y a guère que trois vocabulaires réellement utilisés :

- Celui défini par le WHATWG est hébergé sur microformats.org¹, il reprend presque point à point les vocabulaires les plus célèbres des Microformats. Cela couvre les avis, les pages d'album de musique et d'artiste, les fiches de contact, les événements, les CV, les articles de presse et des produits commerciaux (des boutiques aux petites annonces). C'est très prometteur car tous les lecteurs de Microformat aujourd'hui savent déjà lire cette liste bien établie d'objets, et c'est techniquement très facile de passer de Microformat à Microdata... la mauvaise nouvelle c'est qu'aucun d'eux n'a commencé pour le moment à faire une version Microdata ! C'est quelque part normal puisque Microdata est très peu répandu et est le concurrent officiel de Microformat, il ne reste donc plus qu'à espérer que les choses bougent.
- Un consortium de moteurs de recherche incluant Google, Yahoo! et Bing a créé schema.org² **plusieurs centaines de vocabulaire pour représenter** plusieurs centaines d'objets différents. C'est probablement **votre meilleur pari** car toutes les grandes catégories de sites y sont représentées. Il n'y a par contre pas encore d'outils permettant de valider que votre page est conforme.
- Le vocabulaire défini par Google³ regroupe, lui, des avis, des fiches contact (personnes morales et physiques), des événements, des produits dans des boutiques, des recettes de cuisine, des fils d'Ariane et des vidéos. On peut pratiquement deviner la stratégie de Google rien qu'en lisant cette liste. Le bénéfice immédiat de Microdata est ici bien visible car il permet d'améliorer son affichage dans ce moteur de recherche ultra-dominant et, **à tout le moins**, d'être certain d'être bien compris.

Vous faites un site de cuisine ? Pourquoi ne pas vous démarquer de vos 211 000 camarades avec une petite photo appétissante qui apparaîtrait sur la page de Google ?

1. Voir les fiches qui commencent par un h : <http://microformats.org/profile/>

2. Vocabulaire commun à Google, Bing et Yahoo! : <http://schema.org/docs/full.html>

3. Le site du vocabulaire Microdata de Google : <http://www.data-vocabulary.org/>



Figure 4.1 — Modification du visuel chez Google.

Pour le même prix, le temps de préparation (*Recipe*) et la note ainsi que le nombre d'avis (*Review* et *Review-aggregate*) apparaissent. Bien sûr rien n'est garanti par Google, qui reste maître chez lui, mais au moins vous aurez fait le maximum.

Nous vous renvoyons au site d'aide des webmasters chez Google¹, qui présente des exemples de code bien plus lisibles que les froides spécifications de leur vocabulaire, et qui dispose en plus d'un outil d'analyse des pages vous permettant de tester l'implémentation de Microdata.

webmaster tools

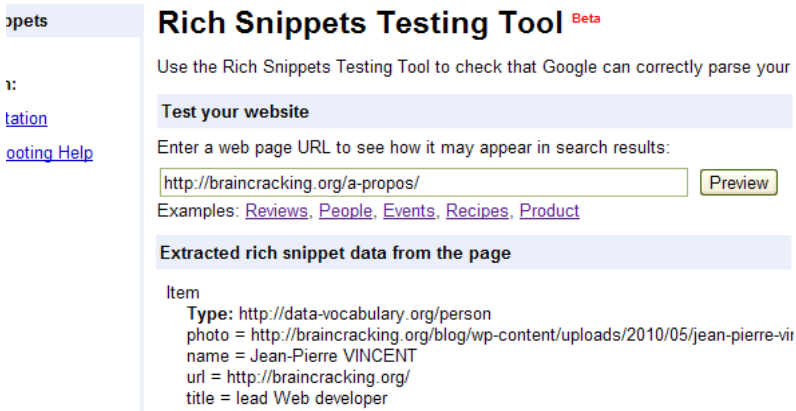


Figure 4.2 — Outil de test Google.

Pratique si vous avez un doute quant à la compréhension de votre page.

1. <http://www.google.com/support/webmasters/bin/topic.py?topic=21997>

En résumé

Microdata possède un avantage immédiat en référencement si l'on utilise le vocabulaire défini les moteurs de recherche. Microformat reste encore le leader de l'enrichissement sémantique des pages car des dizaines de programmes (y compris les robots de Google) savent déjà exploiter les données disponibles. Si les deux formats peuvent cohabiter sur la même page, il semble cependant pratiquement assuré qu'à moyen terme les nombreux vocabulaires de schema.org finiront par convaincre les auteurs de sites.

Notre conseil en 2012 : HTML5 Microdata est supporté par tous les moteurs de recherche significatifs, trouvez-vous un vocabulaire et implémentez-le.

5

ARIA t'emmène au pays de l'accessibilité

Objectif

Passer à HTML5 ne doit pas se faire en oubliant les bonnes pratiques du Web, au premier rang desquelles se place l'accessibilité. HTML5 est accusé d'oublier l'accessibilité des pages web mais fait pourtant beaucoup de référence à WAI-ARIA. En effet, cette spécification du W3C permet de marquer au mieux une page pour aider les utilisateurs de technologie d'assistance, et par ricochet un peu tout le monde. Nous allons donc profiter de ce chapitre pour faire le point sur l'accessibilité naturelle de HTML5, et voir les concepts les plus intéressants de ARIA.

5.1 LE BON RÔLE

5.1.1 Zoner

Le projet WAI-ARIA a analysé les zones fonctionnelles trouvées couramment sur les sites web pour définir les *landmark roles*. Ces zones se marquent avec l'attribut `role` et permettent en théorie de s'affranchir des liens d'évitement qui sont une bonne pratique d'accessibilité reconnue. Souvenez-vous :

Liens d'évitement classiques, pointant vers trois zones importantes de la page

```
../..  
<ul id="skip-link">  
  <li><a href="#menu">Aller au menu</a></li>
```

```

    <li><a href="#content">Aller au contenu</a></li>
    <li><a href="#search">Aller à la recherche</a></li>
  </ul>
  ../..
  <form id="search">
    <label for="term">Chercher des pommes</label>
    <input name="term" type="text" />
    <input type="submit">
  </form>
  ../..
  <div>
    <ul id="menu">
      <li><a href="/">Accueil</a></li>
      <li><a href="/basket">Mon panier de pommes</a></li>
    </ul>
  </div>
  ../..
  <div id="content">
    <p>Nous vendons plusieurs types de pommes
  </div>
  ../..

```

Ces liens se placent souvent avant tout contenu, y compris avant l'en-tête de page. Pour des raisons d'esthétique, on cherche généralement à les cacher de la majorité d'utilisateurs qui navigue à la souris et de manière visuelle.

Note : une méthode reconnue pour les cacher est une couleur de police similaire à la couleur de fond et du JavaScript pour rajouter une couleur visible en cas de focus au clavier. Appuyer sur la touche tabulation les fait donc apparaître.

Le but avoué des rôles de zone ARIA est donc de standardiser cette pratique pour laisser les navigateurs et les lecteurs d'écran cartographier la page et fournir à certains de nos utilisateurs une navigation rapide. Concrètement, dans l'exemple précédent nous allons supprimer les liens et disséminer trois rôles classiques à l'intérieur du code, avec l'attribut `role` sur les bons éléments :

- `search` sur le formulaire de recherche ;
- `navigation` sur le menu, et on remplace la `div` parente par une balise `nav` ;
- `main` sur la `div` entourant le contenu principal du site.

```

Introduction des rôles de zone
../..
<form role="search">
  <label for="term">Chercher des pommes</label>
  <input name="term" type="text" />
  <input type="submit">
</form>
<nav role="navigation">
  <ul id="menu">
    <li><a href="/">Accueil
    <li><a href="/basket">Mon panier de pommes</a>
  </ul>

```

```
</ul>
</nav>
../..
<div id="content" role="main">
  <p>Nous vendons plusieurs types de pommes
</div>
../..
```



Voir la démo !

<http://www.livre-html5.com/aria/region.php>

Saint-Thomas, patron des codeurs s'ils en avaient un, ne croirait que ce qu'il peut voir, et nous suggérerait probablement de vérifier visuellement les zones ARIA des pages. Si Firefox est votre navigateur de développement, alors installez l'addon « Juicy Studio Accessibility Toolbar »¹, qui permet entre autres de bien repérer les zones concernées.

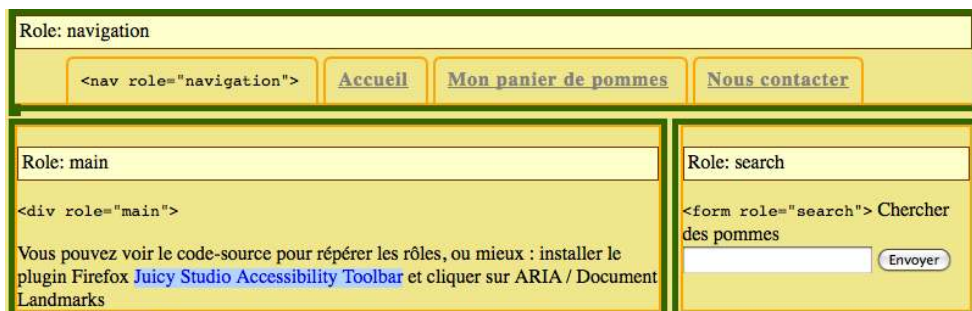


Figure 5.1 — Avec Juicy Studio Accessibility Toolbar,
les zones apparaissent distinctement, ainsi que leur type.

Attention : les liens d'échappement ne deviennent pas totalement obsolètes, malgré les rôles ARIA. Les technologies d'assistance reconnaissent déjà bien ARIA mais il n'y a pas que cette catégorie d'utilisateur qui tirait parti des liens d'échappement : certains internautes naviguent au clavier sans ces logiciels. Cela peut être par habitude (voire snobisme !), pour le côté pratique (les *trackpads* imprécis des *notebooks*, souris sur canapé...) ou par accident (souris sans fil déchargée, enfants farceurs, chat gobeur...). Notre recommandation serait donc : **si vous avez déjà ces liens, ne les supprimez pas car ils sont encore utiles**. Mais dans tous les cas ajoutez les rôles de zone fonctionnelle pour améliorer l'accessibilité.

1. <https://addons.mozilla.org/en-US/firefox/addon/juicy-studio-accessibility-too/>

Il n'existe que huit rôles de zone, nous allons donc les lister ici (tableau 5.1).

Tableau 5.1 — Rôles de zone

Rôle	Unique ou multiple	Balise correspondante	Description
<code>banner</code>	Unique	<code>header</code> , si relative au document	Zone d'informations relatives au site, par opposition à la page. On peut y trouver le logo, le nom du site, une zone de recherche sur le site... Bref elle correspond généralement à l'en-tête du site.
<code>contentinfo</code>	Unique	<code>footer</code> , si relative au document	Zone d'informations complémentaires à la page. On peut y trouver le <i>copyright</i> , les mentions légales et leurs liens... Cela correspond généralement au pied de page du site.
<code>complementary</code>	Multiple	<code>aside</code> , si cette balise est au même niveau hiérarchique que le nœud avec le <code>role="main"</code>	Informations relatives au contenu principal de la page.
<code>navigation</code>	Multiple	<code>nav</code>	Entoure les zones de liens de navigation.
<code>form</code>	Multiple	<code>form</code>	Tous les formulaires, hors recherche.
<code>search</code>	Multiple	<code>form</code>	Les formulaires de recherche. Pensez également à utiliser <code>input type="search"</code> .
<code>main</code>	Unique	-	Le contenu principal de la page.
<code>application</code>	Multiple	<code>body</code> , <code>svg</code> , <code>embed</code> , <code>video</code> ...	À mettre sur les zones contenant une navigation au clavier scriptée (<code>Esc</code> pour fermer une <i>lightbox</i> , flèches pour naviguer ou jouer, gestion des appuis sur <code>Alt</code> ou <code>Ctrl</code> ...) pour demander aux technologies d'assistance de ne rien intercepter.
<code>document</code>	Multiple	<code>body</code> par défaut	C'est le contraire du rôle <code>application</code> , les technologies d'assistance reviennent à leur navigation clavier standard. Exemple : un <i>Webmail</i> devrait se mettre en application mais le corps d'un email devrait repasser en <code>document</code> .

HTML5 spécifie que la position dans l'arbre DOM peut changer la portée des balises. C'est pour cette raison que les rôles ne correspondent pas automatiquement à certaines des nouvelles balises HTML5 : celles-ci peuvent être placées à des niveaux différents et faire référence à la **section parente plutôt qu'au document**. Or les rôles de zone sont toujours relatifs à la page, et non aux sections ; il est donc normal que les rôles finissent parfois sur des `div` contenantes. Voici un exemple où une seule des deux balises `aside` peut prendre le rôle `complementary` :

Seule la seconde balise `aside` concerne la page

```
<article role="main">
  <h1>Un titre bien mystérieux</h1>
  <div>Un article bien court</div>
  <aside>
    <h2>Des commentaires bien absents</h2>
    0 commentaire(s)
  </aside>
</article>
<aside role="complementary">
  <a href="/articles/">Retrouvez d'autres articles tout aussi passionnants</a>
</aside>
```

Le fait d'écrire des choses comme `<nav role="navigation">` est critiquable, car elle fait effectivement écrire deux fois la même chose, ce que déteste tout bon développeur. Certains navigateurs et technologie d'assistance sont déjà capables de faire eux-mêmes le lien, mais pour des raisons de rétrocompatibilité et vu le peu d'effort intellectuel que cela demande, il vaut mieux continuer à coder de cette manière encore quelques années.

5.1.2 Cette zone est vivante

L'ennui avec les applications web, c'est que le DOM ne tient pas en place. Sur les sites statiques de l'ère pré-AJAX, le contenu était chargé en une fois avec le DOM, les utilisateurs n'avaient plus qu'à admirer le travail. Aujourd'hui, certaines zones, voire des pans entiers de contenu, sont entièrement dynamiques et ne sont mises à jour qu'après coup ou régulièrement. C'est le cas des sites affichant des informations boursières mises à jour en continu, des *Webmails* qui doivent signaler l'arrivée de nouveaux messages ou tout simplement du suivi de la progression d'une tâche comme un envoi de fichier ou un encodage de vidéo.

Il y a toutes sortes d'artefacts visuels pour signaler la mise à jour du contenu, mais ceux qui utilisent une technologie d'assistance ne les voient ou ne les remarquent pas. Voici donc `aria-live` qui permet de marquer ces zones. Prenons l'exemple d'un réseau social qui surveillerait continuellement, à l'aide de techniques de « polling » ou de *websocket*, si vous n'avez pas reçu de nouveaux messages et qui mettrait cette information dans une `div`.

Zone d'information sur le nombre de messages en attente

```
<div aria-live="polite">
  Vous avez <span id="number">3</span> messages
</div>
```

Nous avons rajouté l'attribut `aria-live` pour signaler que cette zone est à surveiller. `aria-live` peut prendre trois valeurs :

- `off` : par défaut aucune zone mise à jour après le chargement de la page n'est surveillée ;
- `polite` : lit la zone mise à jour sans interrompre l'utilisateur (recommandé) ;
- `assertive` : interrompt l'utilisateur pour lui signaler la mise à jour.

En choisissant `polite`, nous laissons donc l'utilisateur interagir avec le reste de la page tant qu'il veut, ce qui lui permet par exemple de lire ses mails en entier. Pendant ce temps, un JavaScript met à jour (ou pas) l'élément `span id="number"` avec le nouveau nombre de messages si nécessaire. Il n'y a que lorsque l'utilisateur fera une pause dans sa navigation ou sa lecture que la voix de son lecteur d'écran lui lira la zone et donc lui signalera l'arrivée de nouveaux messages.

L'utilisation de `assertive` doit se faire avec précaution car vous pouvez potentiellement ruiner l'expérience d'utilisation du site si vous mettez à jour fréquemment certaines zones. Vous pouvez l'utiliser pour les informations urgentes comme les messages d'erreur ou un événement qui fait que l'utilisateur devrait arrêter ce qu'il est en train de faire.

Par défaut, les lecteurs d'écran ne lisent que le nœud DOM mis à jour (y compris si son contenu est le même d'ailleurs). Concrètement dans notre cas, **seul le chiffre sera relu**, ce qui n'aide pas forcément beaucoup notre utilisateur. Soit vous changez votre code JavaScript pour vous adapter à ce cas particulier (c'est mal), soit vous utilisez l'attribut `aria-atomic` (c'est mieux).

`aria-atomic` précise la portée de la lecture

```
<div aria-atomic="true" aria-live="polite">  
  Vous avez <span id="number"> 3 </span> messages  
</div>
```

La valeur de `aria-atomic` est ici `true`, je vous laisse deviner l'autre valeur possible, qui est également la valeur par défaut. En mettant cette valeur sur l'élément qui possède l'attribut `aria-live`, c'est tout son contenu qui sera relu et non plus seulement la partie du DOM modifiée.

5.1.3 Les widgets

Les *widgets* sont des ensembles de petites fonctionnalités trouvées fréquemment sur les sites. WAI-ARIA en définit un certain nombre dont les plus communs sont listés dans le tableau 5.2.

Ces rôles sont des raccourcis bien pratiques pour ceux qui ne souhaitent pas s'enfoncer trop avant dans la spécification ARIA. Si ces *widgets* n'existaient pas, voyons par exemple comment nous marquerions un chat (l'application, pas le féliné) :

- `aria-live` pour surveiller les mises à jour de la zone,
- `aria-relevant` et `aria-atomic` pour signifier que seules les dernières lignes ajoutées sont importantes,

- optionnel : `aria-level` pour dire que les lignes font partie du même ensemble de discussion,
- optionnel : `aria-flowto` pour faire comprendre l'ordre d'enchaînement des lignes.

Tableau 5.2 — Widgets ARIA

Rôles	Fonctionnalité
<code>dialog</code> , <code>alertdialog</code>	boîtes de dialogue
<code>log</code> , <code>marquee</code> , <code>status</code> , <code>timer</code>	zones de texte mises à jour en continu
<code>menuitem</code> , <code>menu</code> , <code>menubar</code> , <code>menuitemradio</code> , <code>menuitemcheckbox</code>	menus
<code>scrollbar</code> , <code>slider</code>	barres de défilement non natives, curseurs
<code>progressbar</code>	indicateurs de progression
<code>tabpanel</code> , <code>tablist</code> , <code>tab</code>	onglets
<code>tooltip</code>	tooltips
<code>treeitem</code>	arbres hiérarchisés comme dans un explorateur de fichiers

Heureusement, il y a un rôle approprié : `log` qui a des valeurs par défaut pour tout cela et qui simplifie l'écriture.

Zone de chat marquée façon ARIA

```
<ol role="log">
  <li>KevinDu75 : Kikoo ^__^
  <li>DarkBlackWarrior666 : LOL !!!!!
</ol>
```

Ces rôles sont d'autant plus importants que peu d'entre eux ont une correspondance directe avec des éléments HTML : seul l'auteur de l'application sait que tel paquet de `div` contient une boîte de dialogue, un menu, un chat ou ... rien.

5.2 HTML5 EST-IL ACCESSIBLE ?

La question est sérieusement posée par les experts en accessibilité, et nous allons voir que chaque point mérite discussion pour prendre une décision éclairée.

5.2.1 Les nouveaux éléments posent problème

Les versions d'Internet Explorer antérieures à la 9 construisent un arbre DOM qui ne contient pas les balises inconnues. Non seulement elles ne sont pas stylisables, mais en plus IE ne donne pas l'information aux lecteurs d'écran qui ne voient pas le contenu de ces balises. Pour régler ce problème, il faut exécuter `shim.js`¹ dans le `head`

1. Projet Shim.js : <http://code.google.com/p/html5shim/>

du document, ce qui crée une dépendance envers JavaScript. Si vous ne voulez pas dépendre de JavaScript, vous pouvez travailler vos CSS et votre DOM pour ne jamais avoir à styler ces éléments.

Au moins deux gros bugs d'une combinaison précise du couple navigateur / lecteur d'écran bloquaient l'accès au contenu de certaines des nouvelles balises HTML5. Les chiffres sur le taux de pénétration de ces versions buguées n'existent pas; par contre les seuls chiffres publics¹ sur la vitesse d'adoption des mises à jour chez ces utilisateurs montrent qu'au moins 20 % d'entre eux mettent plus d'un an pour passer à une nouvelle version, ce qui signifie qu'il faut prendre en compte ces bugs pendant plusieurs années.

Pour éviter ces bugs ou la dépendance envers JavaScript, la seule solution viable étant de **ne pas utiliser ces balises**, à vous donc de choisir entre le risque de tomber sur ces versions buggées et le bénéfice à tirer des nouvelles balises.

Parlons alors des bénéfices : outre une meilleure lisibilité du code et peut être un meilleur référencement, beaucoup de progrès en accessibilité ont été réalisés récemment. Firefox pour Windows et Safari pour Mac sont des modèles d'intégration de la nouvelle sémantique, et un grand groupe comme Microsoft a la vocation et les moyens pour suivre ce mouvement.

Au contraire de la première édition de ce livre, on peut conclure aujourd'hui que les nouveaux éléments de sémantique sont à utiliser au même titre que n'importe quel autre élément déjà présent en HTML4.

5.2.2 Le multimédia natif ne tient pas ses promesses

Les éléments **audio** et **video** natifs semblent parfaits pour que le navigateur s'occupe de leur accessibilité, pourtant au moment d'écrire ce livre, les navigateurs gèrent de manière très variée la navigation clavier sur les éléments. Aucun navigateur ne le fait de manière parfaite, certains n'ont même rien prévu du tout et dans tous les cas il n'y a pas de pont spécial entre ces éléments et l'API d'accessibilité exploitée par les technologies d'assistance.

Malgré cela, la navigation reste meilleure que dans l'immense majorité des lecteurs Flash du marché. Ici vous avez donc deux stratégies :

- utiliser uniquement les éléments multimédias natifs et attendre que les navigateurs fassent tout le boulot pour vous ;
- coder votre propre *player* avec lecture de la piste audio/vidéo en Flash si besoin. L'important étant que les contrôles soient faits en HTML pour rester naturellement accessibles.

1. Étude menée par WebAIM en 2010 auprès de 1245 utilisateurs de technologie d'assistance, résultats complets disponibles sur <http://webaim.org/projects/screenreadersurvey3/>

5.2.3 La hiérarchie des titres est cassée

En HTML5, il existe maintenant un algorithme officiel pour déterminer les vrais niveaux hiérarchiques des `Hx`, qui sont maintenant relatifs à la section parente. Concrètement, il y a maintenant dans la source beaucoup plus de titres de niveau 1, puisque le `H1` d'une balise `article` ou `aside` aura en HTML5 la valeur d'un `H2`. Pour visualiser cet algorithme sur une page, il existe un *bookmarklet* nommé HTML5 outliner (`h5o`)¹.

La majorité des navigateurs et des technologies d'assistance n'intègrent pas encore cet algorithme, et vont donc voir beaucoup de titres de niveau 1 tandis que les navigateurs avec un moteur HTML5 vont réinterpréter la hiérarchie des titres de vos sites, même si vous ne passez pas au nouveau *doctype*.

La navigation par titre étant le principal moyen pour ces utilisateurs de découvrir une page, cette expérience est donc importante. D'un autre côté très peu de sites ont une hiérarchie cohérente, et si il n'y a pas trop de titres sur la page, tout avoir au premier niveau n'est pas très dérangeant.

En conclusion, deux stratégies :

- si vos hiérarchies de titre de page étaient bien calibrées, ne changez rien ! le `H2` d'une section sera toujours un titre de niveau 2 pour la page ;
- si jusqu'ici vous ne gériez pas les niveaux de titre correctement sur l'ensemble de la page, passer à la logique HTML5 vous permettra justement de les gérer plus facilement, module par module. Les navigateurs, les technologies d'assistance et probablement les moteurs de recherche vont s'adapter peu à peu.

La majorité des sites sont probablement dans le cas où une hiérarchie bien organisée de titres n'est pas nécessaire, cependant certains sites à fort contenu textuel (livres, encyclopédies, longs articles...) préféreront peut-être sacrifier un peu de sémantique pour conserver une navigation accessible.

5.2.4 Canvas n'est pas accessible

Effectivement, la zone `canvas` est pour les technologies d'assistance un trou noir d'où rien ne sort, ce qui est le reproche que l'on faisait déjà à Flash. Ironiquement d'ailleurs, Flash est même plus accessible que `canvas`, pour peu que le développeur flash active les options d'accessibilité et prévoie son application dans ce sens (ce qui est rarement le cas...). Il y a un lointain espoir que cela change, car un groupe de travail étudie la possibilité de rajouter un DOM à Canvas (un peu comme SVG), et une implémentation en a été réalisée dans Internet Explorer 9. Cependant cela reste expérimental.

1. Le bookmarklet HTML5 outliner : <http://code.google.com/p/h5o/>

Conclusion

Ressortez donc les bonnes pratiques et fournissez un contenu alternatif lorsque cela a du sens : si vous utilisez Canvas pour des effets décoratifs, c'est inutile. Si vous l'utilisez pour de la navigation, vous vous êtes probablement trompé de technologie, mais vous devez fournir une alternative comme vous le faisiez pour Flash. Si c'est le cœur de votre application (un jeu, une application de manipulation d'image...) il nous semble difficile de fournir une alternative, donc à défaut un petit mot d'excuse suffira.

Comment tester l'accessibilité ?

Pour tester ARIA, il vous faut un Windows, un Firefox et un Internet Explorer 8 qui sont les deux navigateurs les plus utilisés par les clients des lecteurs d'écran. Ce sont également les deux navigateurs qui supportent le mieux ARIA. En lecteur d'écran **JAWS**¹ par Freedom Scientific est clairement le leader du marché, suivi par **Window-Eyes**² de GW Micro. Ils offrent tous deux des versions d'essai gratuit dont la limite est que vous devez redémarrer Windows toutes les 40 minutes. Voici donc l'astuce de sioux : utilisez des **machines virtuelles**, et la fonctionnalité *snapshot* pour vous éviter ces redémarrages fastidieux ou de payer des licences à plusieurs milliers de dollars. Et reversez-nous 10 % de l'économie réalisée, d'avance merci.

Vous pouvez également utiliser un lecteur d'écran open source : **NVDA**³ ainsi que la *toolbar* « Juicy Studio Accessibility Toolbar »⁴ pour Firefox qui permet de vérifier ses zones pendant le développement.

5.3 L'ACCESSIBILITÉ NATURELLE DE HTML5

L'avantage de la standardisation des fonctionnalités et balises, c'est que les navigateurs vont avoir une base stable pour rajouter automatiquement une couche d'accessibilité à vos pages. En fait, les spécifications ARIA et HTML5 se sont synchronisées pour faire une correspondance entre des propriétés ARIA et certains éléments. Nous avons vu l'opportunité offerte par les balises `header`, `nav` ou `footer` sur lesquelles les navigateurs peuvent rajouter automatiquement des rôles de zone, voici quelques autres exemples où HTML5 pourra être accessible sans que le développeur y pense.

5.3.1 Zone de l'écran

Certains navigateurs commencent d'ores et déjà à établir la correspondance entre certaines balises et les rôles de zone, telle que définie à la figure 5.1. En attendant le jour béni des dieux du Web où tous les navigateurs feront de même, notre recommandation

1. Site web de JAWS : http://www.freedomscientific.com/fs_products/software_jaws.asp

2. Site de Window-Eyes : <http://www.gwmicro.com/Window-Eyes/>

3. Site de NVDA : <http://www.nvda-project.org/>

4. Téléchargeable ici : <https://addons.mozilla.org/en-US/firefox/addon/juicy-studio-accessibility-too/>

reste de continuer tout de même à ajouter les attributs à la main, comme dans le fameux `<nav role=navigation >`.

5.3.2 Légende et figure

Vieux serpent de mer, pouvoir associer une légende à une image ou un tableau devient enfin réalité dans HTML5, avec les éléments `figure` et `figcaption`. Parallèlement ARIA a la propriété `aria-labelledby` qui est plus générique, mais qui dans le cas d'une image signifie précisément la même chose. Pour l'instant seul Firefox Windows a implémenté la relation entre les deux, mais en attendant les autres voici à quoi ressemble le code :

Ici nous définissons une image avec sa légende, en HTML5 et en ARIA

```
<figure aria-labelledby="image-42">
  
  <figcaption id="image-42"> Photo prise avant l'impact </figcaption> </figure>
```

Notez que comme il est d'usage l'attribut `alt` décrit l'image pour que le lecteur s'en fasse une représentation mentale (en ce moment même vous devriez avoir le refrain de « chabada » en tête). `figcaption` par contre donne un titre qui s'intégrerait à un récit illustré par l'image. Ici le récit pourrait être un article de blog relatant des vacances à la plage se terminant par deux nez cassés.

5.3.3 Barre de progression

Un des *widgets* reconnu par ARIA est un indicateur de progression. Voici comment il fallait marquer une barre de progression en ARIA pour qu'elle soit compréhensible :

- déclarer la zone qui sera mise à jour avec `aria-live`. La valeur `polite` permet de ne pas interrompre l'utilisateur s'il lit une autre zone de l'écran ;
- marquer l'élément de la barre de progression avec le rôle de widget `role="progressbar"` ;
- déterminer la plage des valeurs numériques possibles avec `aria-valuemin` et `aria-valuemax` ;
- mettre à jour la valeur de la progression avec `aria-valuenow`.

Code d'une barre de progression classique, enrichie avec ARIA

```
// restons polite
<div id="progress-bar" aria-live="polite">
  <span role="progressbar"
    aria-valuemin="0" aria-valuemax="100"
    aria-valuenow="0"> 0% </span>
</div>
<script>
// On agit sur l'élément enfant
var oProgress = document.getElementById('progress-bar').firstChild;
// appelée à chaque mise à jour
```

```
function updateProgress( iPercent ) {  
    // mise à jour du texte lisible  
    oProgress.innerHTML = iPercent+'%';  
    // mise à jour de l'affichage  
    oProgress.style['width'] = iPercent+'%';  
    // mise à jour ARIA  
    oProgress.setAttribute('aria-valuenow', iPercent);  
};  
</script>  
<style>  
/* la barre de progression n'est qu'un bloc à fond vert  
* dont on met à jour la largeur  
*/  
#progress-bar span {  
    display:block;  
    height:20px;  
    width:0%;  
    background-color:green;  
    color:white;  
}  
/* conteneur de la barre de progression */  
#progress-bar {  
    width:200px;  
    height:20px;  
    border:1px solid black;  
    background-color:white;  
}  
</style>
```



Vous pouvez voir une implémentation simple de barre de progression « à l'ancienne » à cette adresse :
<http://www.livre-html5.com/balises/progress-meter.php>

Notez bien : si vous utilisez un *widget* JavaScript (jQueryUI, YUI ou autre) qui affiche une barre de progression et que votre site est destiné à tous les navigateurs, vérifiez bien qu'il gère pour vous les propriétés ARIA. Si vous devez le coder vous-même, vous pouvez mettre votre barre de progression à l'intérieur du code de la balise `progress` : seul les navigateurs supportant la balise l'afficheront.

HTML5 aussi définit un indicateur de progression natif, le code s'en trouve dès lors simplifié :

- pas de CSS ou presque puisque la barre est stylée nativement,
- une seule valeur à mettre à jour en JavaScript,
- HTML simple : une seule balise.

Code d'une barre de progression native

```
<progress value="0" min="0" max="100" id="progress-bar"></progress>
<script>
// appelée à chaque mise à jour
function updateProgress( iPercent ) {
    // Une seule valeur à mettre à jour
    document.getElementById('progress-bar').value = iPercent;
};
</script>
<style>
/* Même sans CSS c'est joli à voir ! */
</style>
```

Actuellement Chrome et Firefox Windows répercutent déjà les mises à jour sur l'interface d'accessibilité qu'ils présentent aux technologies d'assistance, le développeur web peut donc se concentrer sur le code métier autour de cette barre de progression (ou prendre une pause-café).

5.3.4 Formulaires

Beaucoup des propriétés et rôles ARIA font référence à des éléments déjà existants en HTML4 :

- `link` pour `<a href="">`,
- `button`, `checkbox`, `radio`, `option` pour leur équivalent HTML,
- `textbox` pour `<input type="text">`,
- `aria-labelledby` qui est le symétrique de `<label for="">`,
- `aria-multiline` pour `textarea`,
- `aria-readonly` et `aria-disabled` pour `<input disabled>`,
- `aria-selected` pour `input selected` et `<input checked>`.

Ils sont là au cas où vous n'utiliserez pas les éléments natifs, ce qui est généralement déconseillé, mais qui vous permet plus de liberté graphique.

HTML5 lui introduit de nouvelles notions natives qui correspondent à d'autres rôles et états :

- `range` pour `<input type="range">` (d'ailleurs ce champ est plutôt bien géré pour la navigation au clavier),
- `aria-invalid` sur les `input` mal renseignés,
- `aria-required` pour `<input required>`,
- `aria-valuenow`, `aria-valuemin` et `aria-valuemax` pour les attributs `min`, `max` et `value`.

L'attribut `placeholder` est lui aussi prometteur pour donner plus d'informations à tous les utilisateurs sur ce que l'on peut mettre dans un champ, mais les premières implémentations navigateur commencent déjà à diverger ! Ce n'est pas comme si nous n'étions pas habitués à ce genre de conflit, et la bonne nouvelle c'est qu'il reste recommandé de commencer à l'utiliser immédiatement.

On peut également s'attendre à ce que les navigateurs fassent le travail très complexe de rendre accessible des *Widgets* avancés tels que les différents sélecteurs de date et d'heures (`input type="date"`, `time`, `datetime`, `week` ...). Ceux-ci sont aujourd'hui faits en JavaScript par des bibliothèques ou pire par chaque développeur, et l'accessibilité est rarement prise en compte.

5.3.5 Autres gadgets

HTML5 introduit `details`, qui est une interaction classique du Web : dévoiler un contenu lors du clic ou au survol d'un élément. Il n'y a actuellement pas d'implémentation de cette balise, mais il y a une opportunité pour les navigateurs de mettre correctement à jour la propriété `aria-expanded` en la mettant à `true` ou `false` selon l'état de la balise `details`.

L'auto-complétion est aussi un classique du Web : sous un champ texte, des valeurs sont proposées pour chaque nouvelle lettre tapée par l'utilisateur. En HTML5, la balise `datalist` couplée à un champ `<input type="text" list="list">` va permettre nativement cette fonctionnalité (implémenté aujourd'hui par Opéra). En ARIA, il faut jouer avec les propriétés `aria-autocomplete`, `aria-haspopup` et enfin `aria-owns`, ce que les navigateurs commencent à faire pour vous.

L'édition en ligne activable au clic (lorsque l'on clique sur le titre d'une photo par exemple), est un *widget* JavaScript assez souvent maltraité par les bibliothèques du point de vue de l'accessibilité. Avec `contentEditable="true"` (inventé par IE6, standardisé dans HTML5), les navigateurs et les technologies d'assistance ont déjà pris sur eux de vous faciliter la tâche et de gérer les problèmes de `focus` tout en avertissant l'utilisateur que l'élément est éditable.

En résumé

Quelle stratégie adopter si vous voulez faire rimer HTML5 avec accessibilité ? Cela dépend de votre point de départ :

- Si vous avez déjà implémenté de l'ARIA sur vos pages vous pouvez rajouter des balises HTML5 en complément, pour les autres bénéfices que cela apporte.
- Si vous n'aviez rien et que faire ce double travail de balisage HTML5 et d'attributs ARIA ne vous gêne pas, vous aurez le meilleur des deux mondes immédiatement.
- Si vous n'aviez rien et que lire la spécification ARIA vous rebute, faites uniquement le travail de balisage HTML5 en faisant confiance aux navigateurs pour faire la traduction en ARIA. Après tout les constructeurs de navigateurs ont lu la spécification, EUX.

DEUXIÈME PARTIE

Les applications web : les API

Le second volet de HTML5 est probablement le plus innovant. C'est à partir de là que les développeurs web commencent à changer de métier et à faire de la programmation qui ressemble de plus en plus à du logiciel. C'était également ce qui intéressait le WHATWG et qui a motivé la scission avec le W3C.

Nous allons voir dans cette partie toutes les nouvelles API dignes d'intérêt, implémentées de manière équivalente par au moins deux navigateurs, avec une indication sur la manière de faire marcher la fonctionnalité sur d'autres navigateurs.

Certaines de ces API comme la géolocalisation ou les *WebSockets* ont été sorties de la stricte spécification HTML5 du W3C, mais n'en restent pas moins des standards, avec un support navigateur encourageant.

6

Audio et vidéo

Objectif

Dans son effort de standardisation, le W3C ne pouvait pas oublier une composante aujourd'hui banale des sites web : la vidéo et l'audio. Au cours des années 2000 ce domaine a été le théâtre sanglant d'une bataille entre *plugins*, dont Flash est sortie grand vainqueur. HTML5 va essayer de ravir cette position dominante, mais nous allons voir que si le futur est radieux, le présent l'est un peu moins.

Au programme donc : utilisation simple et avancée des éléments natifs, comment les mettre en production avec ce que cela implique en terme de *codec* et de compatibilité.

6.1 LES ÉLÉMENTS NATIFS

Le bonheur c'est simple comme une balise. En théorie insérer une vidéo ou du son dans une page n'est pas plus compliqué que pour les images.

Simple is beautiful

```
<video src="video.url"></video>  
<audio src="boum.wav"></audio>
```

Avec ça, vous avez une vidéo. Elle ne se joue pas, elle n'est pas contrôlable par l'utilisateur, n'est pas compatible avec tous les navigateurs, ne propose pas de méthode alternative. Mais vous affichez déjà la première image et ce dans la bonne dimension. Côté son ça n'est pas mieux, vous avez simplement fait télécharger les métadonnées du fichier.

Voyons comment contrôler tout cela.

6.1.1 Le markup

Construisons un *player* classique avec un peu plus d'attributs.

Un peu plus d'options

```
<video src="video.url"
  controls
  width="500"
  poster="image.png"
  loop
>
  Votre navigateur est démodé
</video>
<audio src="audio.wav"
  controls
  loop
>
  J'insiste : votre navigateur a vraiment fait son temps
</audio>
```

Notez que tout HTML (ici une simple phrase) inclus entre les balises ouvrantes et fermantes ne sera interprété que par les navigateurs qui ne connaissent pas les balises de média. Concrètement c'est ici que vous mettrez votre lecteur Flash ou une phrase d'excuse.

Passons en revue chaque attribut.

controls

La présence de cet attribut vous permet d'afficher la barre de contrôle multimédia natif du navigateur, ce qui selon les couples OS / navigateur donne ce qui suit.



Figure 6.1 — Les implémentations des barres de contrôle natives audio et vidéo des navigateurs principaux.

Il est normal que ces contrôles ne se ressemblent pas et il est en tout cas souhaitable d'un point de vue ergonomique que l'utilisateur retrouve une barre de navigation à laquelle il est habitué.

Tous les navigateurs ont choisi de mettre ces barres par-dessus la vidéo, et de les faire disparaître lorsqu'elles ne sont pas actives. Pour le développeur c'est assez pratique car il n'a pas à additionner la hauteur de la barre de contrôle à la hauteur de la vidéo qu'il ne connaît d'ailleurs pas forcément.

Le plus des contrôles natifs, c'est également l'accessibilité : seul IE passait automatiquement le *focus* clavier à Flash, ce qui fait que la moitié des utilisateurs n'avaient pas de contrôle clavier sur les vidéos. Et pour les utilisateurs de IE, encore fallait-il que le *player* Flash ait été prévu pour gérer les accès au clavier ce qui est rarement le cas des librairies. Avec des contrôles natifs, les navigateurs peuvent satisfaire naturellement cet usage, voire donner des informations aux technologies d'assistance. Comme tout n'est pas rose, voir le paragraphe 5.2.2 sur l'accessibilité à ce sujet.

width, height

Comme les images, cette paire d'attributs permet de définir hauteur et largeur de la zone de vidéo, et s'exprime en pixel (utilisez les CSS pour une autre unité). Cela n'a par contre pas le même effet que pour les images ou que pour les *players* Flash :

- Si les deux attributs existent, la vidéo n'est pas déformée, elle est simplement rétrécie pour rentrer dans le cadre, en conservant le ratio d'origine. À noter qu'avec un simple style `background-color:black` sur l'élément suffit pour faire des bandes noires.
- Si seul `width` ou `height` est défini, le comportement est le même que pour les images : la dimension est bien appliquée, et le ratio du média détermine l'autre dimension.
- Si rien n'est défini, les dimensions sont :
 - celles de la vidéo,
 - à défaut les dimensions de l'image dans `poster`,
 - à défaut 300 par 300 pixels.

L'élément `audio` n'est pas censé supporter ces attributs, par contre vous pouvez définir ses dimensions en CSS, il aura alors le comportement d'un élément de type `inline-block`. Si la barre de contrôle est visible elle s'étirera en largeur.

poster

Il s'utilise comme le `src` d'une balise `img` : on lui donne une URL pointant vers une image et il l'affiche tant que la vidéo n'est pas démarrée. Très pratique car le comportement par défaut de l'élément `video` est d'aller chercher la première image du flux vidéo, ce qui correspond souvent à une image noire, ou au mieux une partie non représentative de la vidéo.

Suggestion : pour faire un effet graphique en mettant un gros bouton « lecture » au-dessus de la vidéo, pas la peine de l'encoder dans l'image du poster. Placez cet élément graphique simplement au-dessus de la vidéo en CSS.

loop

En audio comme en vidéo, permet de jouer en boucle le média.

Astuce : si vous voulez irriter vos visiteurs et les faire quitter votre site, nous vous suggérons une combinaison de `autoplay` et `loop`. Seconde astuce : il y a parfois de l'ironie dans ce livre, nous comptons sur votre sagacité.

6.1.2 Contrôle du flux

Maintenant que nous maîtrisons ce que l'utilisateur voit, penchons-nous un peu sur l'expérience utilisateur et l'utilisation qui va être faite du réseau. En effet comme pour les images sur les modems d'il y a 15 ans, l'utilisation des fichiers multimédia de par leurs poids n'est pas anodine, et cela est encore plus vrai sur mobile. Et il n'est pas forcément agréable d'entendre une piste audio se lancer sans que l'on sache d'où elle provient.

autoplay, preload

Ces deux attributs sont valables pour `audio` et `video` et ont les mêmes effets.

`autoplay` permet vous vous en doutez de jouer le média sans demander son avis à l'utilisateur, ce qui peut être pratique pour une page avec une vidéo isolée ou une web radio. Certains navigateurs mobiles ignorent cette directive.

`preload` lui prend trois valeurs :

- `auto`, la valeur par défaut, qui laisse le choix au navigateur d'estimer si son utilisateur n'a que faire de gaspiller sa bande passante (les navigateurs de bureau ne jouent pas ça aux dès, ils regardent la configuration utilisateur tandis que les mobiles ne pré-chargent rien) ;
- `none`, qui permet au propriétaire du site de ne pas gaspiller sa bande passante pour des vidéos qui ne seront pas forcément lues ;
- `metadata`, qui permet d'aller chercher la première image et les dimensions d'une vidéo, et d'afficher la durée totale du média.

Vous l'aurez compris, si vous payez votre trafic ou par souci d'écologie (chaque bit qui passe a une signature carbone, sachez-le !), spécifiez `preload` avec les valeurs `none` ou `metadata`.

Note : la spécification ayant changé, Firefox avant la version 4 reconnaissait l'attribut `autobuffer` dont la présence était l'équivalent de `preload=auto` et l'absence signifiait `none`.

Au final il y a quatre choses qui vont déclencher un aller / retour serveur pour aller chercher les métadonnées de la vidéo :

1. `preload="metadata"`, et encore heureux, c'est ce qu'on lui demande. Cela marche aussi pour l'élément `audio` ;
2. l'absence de `poster`, car l'élément `video` va aller chercher la première image de la vidéo ;
3. l'absence totale ou partielle de `height` et `width`, car `video` essaye de déterminer ses dimensions à partir de la première image de la vidéo ;
4. l'absence de l'attribut `type` dans un élément `source`.

Les media queries

Admettons que vous ayez tout réglé pour qu'aucun bit ne transite sur le réseau inutilement, voici maintenant que l'utilisateur appuie sur le bouton « play ». Très bien, mais quelle vidéo lui servir ? Une vidéo HD réglée en 720 p ou une modeste version en 480 × 360 ?

Les navigateurs peuvent être capables de choisir eux-mêmes le fichier le mieux approprié à l'appareil sur lequel ils s'exécutent, pour peu que vous leur précisiez quelle source aller chercher pour quel cas. Pour définir plusieurs sources alternatives, il faut enlever l'attribut `src` et rajouter plusieurs éléments `source`.

La balise source

```
<video controls>
  <source src="video.hires.url"></source>
  <source src="video.lowres.url"></source>
</video>
```

Le navigateur sait maintenant qu'il a plusieurs options, ce qui lui fait une belle jambe s'il ne sait pas sur quel critère choisir. Arrive donc l'attribut `media` qui est le même que l'attribut `media` de la balise `link` ou du sélecteur CSS3 `@media`.

Faisons simple : vous n'avez qu'une version basse et une haute résolution de votre vidéo, et vous voulez envoyer la basse résolution à tous les mobiles.

Cibler les mobiles

```
<video controls>
  <source src="video.lowres.url" media="handheld"></source>
  <source src="video.hires.url"></source>
</video>
```

Si vous connaissez déjà les *media queries*¹, vous savez sûrement que vous pouvez aller beaucoup plus loin et carrément cibler en fonction de certaines caractéristiques physiques de la machine. Votre site s'affiche sur une télévision connectée et vous avez une version 4/3 et 16/9 de votre vidéo ?

1. Documentation en français sur les *media queries* sur le Mozilla Developer Center, https://developer.mozilla.org/fr/CSS/Media_queries.

Cibler par ratio d'écran

```
<video controls>
  <source src="comescope.url"
    media="device-aspect-ratio : 4/3"></source>
  <source src="cinemascope.url">
    media="min-device-aspect-ratio : 16/9"></source>
</video>
```

Cela sera surtout utile pour détecter les petites résolutions d'écran, qui correspondent généralement aux mobiles, afin de servir des vidéos de plus basse qualité et donc moins lourdes. Une moins bonne qualité ne devrait pas se voir sur ces petits écrans. Voici un exemple de syntaxe.

Cibler par résolution d'écran

```
<video controls>
  <source src="video.hires.url"
    media="(min-device-width :800px)"></source>
  <source src="video.lowres.url"></source>
</video>
```

Ici on distribue la vidéo haute résolution aux écrans qui font au moins 800 pixels de large, et une vidéo basse résolution pour tous les autres. On parle bien des pixels physiques, et non pas de *viewport* navigateur, de la résolution d'écran de l'OS ou de la largeur de la balise `video`. Cela est pour cibler le seul cas où une vidéo de mauvaise qualité se voit : le plein écran.

La taille de 800 pixels n'est pas si arbitraire que cela, elle permet de se placer au-dessus des tailles d'écran généralement constatées sur les *smartphones* et tablettes les plus utilisées sur le Web.

Note : les *media-queries* n'ont pas prévu de cibler par débit, ce qui dans le cas de la vidéo aurait été vital. Certains bricolages à base de JavaScript, de téléchargement d'image et de cookie peuvent suffire mais sont assez fragiles. Sinon certains serveurs de streaming vidéo et audio savent s'adapter au débit constaté.

6.2 L'API JAVASCRIPT

Une fois votre élément `audio` ou `video` créé, vous avez accès via JavaScript à 22 événements, 27 propriétés, 9 constantes et 5 méthodes. Épargnons quelques arbres : nous n'allons parler que de certaines parties de l'API en fonction de votre besoin. La liste complète est bien sûr consultable dans la spécification¹ et sur une page de démonstration bien pratique du W3C².

1. API des éléments multimédia : <http://www.w3.org/TR/html5/video.html#media-elements>

2. Démo de l'élément vidéo : <http://www.w3.org/2010/05/video/mediaevents.html>

6.2.1 Tester le support

Commençons d'abord par savoir si le navigateur supporte les éléments multimédias natifs, afin d'afficher votre lecteur Flash si ce n'est pas le cas. Nous avons vu dans le premier exemple du paragraphe 6.1.1 Le markup que vous pouvez le faire en HTML, voici l'équivalent en JavaScript.

Détection du support navigateur

```
// version longue
function doesSupport() {
    var oElement = document.createElement('video');
    if(oElement.canPlayType)
        return true;
    else
        return false;
}
// version courte
(!document.createElement('video').canPlayType); // true ou false
```

Comme beaucoup de test de support HTML5, il se base sur la capacité du navigateur à créer avec un nouvel élément les nouvelles méthodes qui vont avec, ce qui évite le *browser-sniffing*. Si la valeur est `false`, à vous d'insérer dynamiquement votre lecteur traditionnel.

Mais dans le cas du multimédia, savoir que l'élément natif est supporté ne suffit pas, encore faut-il savoir si le format du flux audio ou vidéo est supporté. Référez-vous au paragraphe La trilogie de la guerre des codec's pour plus de détails, mais admettons que vous ayez choisi de ne distribuer votre vidéo qu'au format H.264 ou votre son uniquement au format MP3. Opéra, Chrome et Firefox, supportant les nouvelles balises, vont répondre positivement au code précédent mais ne pourront pas lire pour autant votre vidéo nativement. Il n'y a qu'en JavaScript que vous pouvez demander au navigateur s'il sait lire le format de la vidéo. Cela se fait avec la fonction `canPlayType` et sa réponse peut surprendre si l'on n'est pas préparé.

Détection du support des formats

```
function doesSupportMP4() {
    var oElement = document.createElement('video'),
        sAnswer = oElement.canPlayType('video/mp4');
    switch(sAnswer) {
        case '':
        case 'maybe':
            return false;
        case 'probably':
            return true;
    }
}
```

Si vous vous attendiez à un booléen, c'est loupé. La fonction `canPlayType` peut renvoyer trois valeurs :

- '' ou chaîne vide si vous préférez, ce qui veut dire « ne se prononce pas », et qui correspond normalement à un non ;

- `probably` qui correspond normalement à un oui ;
- `maybe`, qui exprime bien toute la complexité du domaine des formats d'encodage : soit vous êtes optimiste et vous estimez (ou avez constaté sur tous les navigateurs ET les OS) que votre format de vidéo et de son est lisible partout, soit vous êtes prudent et vous ne voulez pas faire jouer une vidéo sans le son (ou le contraire), auquel cas comme ici `maybe` signifie non

Le type mime MP4 ne veut pas dire grand-chose car c'est un conteneur autour d'un codec vidéo (H.264 dans divers formats) et un codec audio (MP3, AAC dans divers formats aussi), c'est pour cela que le navigateur ne s'avance pas trop.

Pour éviter d'avoir la réponse `maybe`, il faut poser la bonne question et inclure en plus du type mime du format conteneur, les formats son et vidéo dans lesquels sont encodées vos vidéos. Ici à la place de `video/mp4`, on aurait du mettre quelque chose comme `'video/mp4; codecs="avc1.42E01E, mp4a.40.2"'`.

Si votre navigateur vous répond qu'il ne supporte pas cette configuration, vous pouvez insérer dynamiquement votre lecteur Flash.

6.2.2 Remplacer les contrôles natifs

Nous vous encourageons à laisser les éléments natifs pour des raisons d'ergonomie et d'accessibilité (un jour), cependant vous souhaitez peut-être créer un lecteur vidéo réellement accessible à tous, ou bien avec le design de votre site, et qui ait accessoirement le même design pour la version Flash. Il est aussi possible que vous utilisiez l'élément `video` comme élément de décoration, ou que vous fassiez une Web Radio à partir de l'élément `audio`, en d'autres termes, vous voulez fabriquer votre propre lecteur de vidéo, auquel cas voici les étapes importantes à prendre en compte.

C'est prêt ?

À partir de quand un fichier multimédia est-il jouable sans à-coups ? D'abord il faut vous assurer que vous et le navigateur connaissiez la longueur totale de la vidéo, et pour cela il y a les métadonnées (voir plus haut dans la partie « autoplay, preload » ce qui déclenche leur chargement). Si elles ne sont pas chargées à cause du markup, vous pouvez aider un peu en lançant la fonction `play()`. Il vous faut alors écouter l'événement `loadedMetadata`.

Attendre les métadonnées

```
<video id="ma-video" src="video.url"></video>
<script>
var oVideo = document.getElementById('ma-video') ;
// au cas où le navigateur ne soit pas allé chercher tout seul les données
if(oVideo.readyState < oVideo.HAVE_METADATA)
    oVideo.play() ;
//
oVideo.onloadedmetadata = function( ) {
    oVideo.duration ; // durée totale en secondes
    oVideo.readyState ; // entier entre 0 et 4, voir les constantes
```

```
oVideo.videoWidth ; // réelle largeur de la vidéo
};
</script>
```

Notez que nous avons utilisé la propriété `readyState` de la vidéo, ainsi qu'une des constantes correspondantes pour connaître l'état de réception des métadonnées. Une fois que vous avez les métadonnées, vous pouvez mettre à jour votre interface avec le temps total de la vidéo. Reste à savoir si vous voulez la jouer tout de suite ou pas.

Si vous voulez afficher un indicateur de chargement, il va vous falloir écouter certains événements et bien les choisir :

- écoutez `canplay` si vous voulez que la lecture commence le plus tôt possible, même si le débit est insuffisant. Pour `readyState`, cela correspond à la constante `HAVE_FUTURE_DATA` ;
- écoutez `canplaythrough` si vous ne voulez pas d'à-coups dans la lecture de votre média. Le navigateur se charge d'estimer le débit et la durée de la vidéo et vous prévient quand il pense qu'il en a déjà suffisamment récupéré. Pour `readyState`, cela correspond à la constante `HAVE_ENOUGH_DATA`.

Faire son interface

Si vous gérez votre propre barre de progression de lecture et de téléchargement, trois propriétés et trois événements vous intéressent :

1. la longueur totale, exprimée en secondes dans l'attribut `duration`. Mis à jour après l'événement `loadedMetadata` que nous venons de voir ;
2. les parties téléchargées, dans l'attribut `buffered` qui contient deux fonctions (`start` et `end`) renvoyant les limites. Mis à jour avec l'événement `progress` ;
3. l'endroit où se trouve la tête de lecture, dans l'attribut `currentTime`, mis à jour avec l'événement `timeUpdate`.

Petit exemple concret, qui part du principe que nous connaissons déjà la durée du média.

Calcul de la progression du téléchargement et de la lecture

```
var oVideo = document.getElementById('ma-video') ;
// calcul du pourcentage du fichier téléchargé
oVideo.onprogress = function() {
    var iBuffered = oVideo.buffered.end( 0 ) ;
    console.log( iBuffered / oVideo.duration ) ;
} ;
// calcul de la position de la tête de lecture
oVideo.ontimeupdate = function() {
    console.log(oVideo.currentTime / oVideo.duration ) ;
} ;
```

Une fois que l'on a nos calculs de pourcentage, l'interface peut être mise à jour, voici par exemple ce que cela donnerait avec un élément `progress` pour le téléchargement et l'élément `input type=range` pour afficher la tête de lecture.

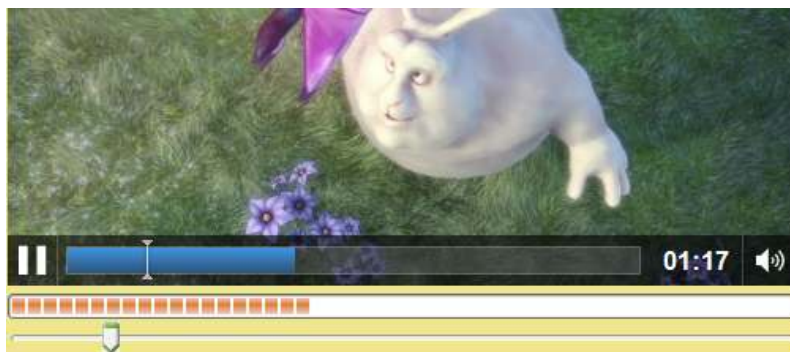


Figure 6.2 — Barre de progression avec éléments natifs sous Chrome XP.



Voir la démo !

<http://www.livre-html5.com/video/progress.php>

Vous pouvez bien sûr enlever les contrôles natifs et les remplacer par des boutons de lecture HTML, il suffit de lancer les méthodes `play()`, `pause()` ou `mute()` de l'objet `video` ou `audio`. Pour contrôler le volume ou se déplacer dans la vidéo il suffit de redéfinir les propriétés `volume` (entre 0 et 1) et `currentTime`.

Note : si vous devez utiliser Flash comme moyen de lecture, il est recommandé de définir vos propres boutons en HTML, qui seront plus accessibles que Flash, et qui vous permettront d'avoir une interface similaire sur tous les navigateurs. Flash fournit nativement les mêmes événements que HTML5, mais il faut coder en ActionScript un pont de communication entre Flash avec JavaScript.

6.3 LES CHOSES QUI FÂCHENT : LES FORMATS

Vous aurez remarqué que dans les exemples précédents, nous ne donnons aucune indication du format possible de la source vidéo ou audio. Vous allez voir que le domaine de la vidéo est assez complexe

6.3.1 La trilogie de la guerre des codecs

L'avènement de l'empire H.264

Dans un Web pas si lointain, chaque grande compagnie a essayé d'imposer son format de vidéo, entendez par là un encodage et un lecteur. Les investissements en ressources

scientifiques et en ingénierie pour créer des codecs sont énormes, et les retombées commerciales étaient intéressantes car avec le codec venait normalement un plugin externe, qui permettait via son installation sur l'ordinateur d'avoir accès à l'utilisateur. Parmi les compétiteurs, citons Apple Quicktime, Microsoft Media Player, Real Player, Adobe Shockwave, VRML, VR...

Pour des raisons qui n'ont rien à voir avec la technique, l'empereur H.264 s'est imposé sur le Web comme ailleurs face à ses concurrents commerciaux, via son chevalier noir : Flash. Il existe une petite résistance rebelle *open-source* qui essaye de survivre : Theora.

H.264 est exploité par le consortium MPEG-LA, dont des dizaines de membres perçoivent des royalties sur l'exploitation du codec (un décodeur inclus dans un navigateur coûte cinq millions de dollars par an par exemple). Pour ce qui concerne le Web ses membres les plus connus sont Microsoft et Apple ce qui explique que leurs navigateurs respectifs ne supportent nativement que H.264.

En quoi cela concerne les développeurs web ? La licence MPEG-LA est une épée de Damoclès (laser) au-dessus des sites qui diffusent des vidéos : aujourd'hui ceux qui « tirent des revenus directs des vidéos distribuées »¹ doivent reverser des royalties. La formulation elle-même est floue (afficher de la pub à côté ou dans une vidéo rentre-t-il dans ce cas ?), mais en pratique aujourd'hui seuls ceux dont le métier est la *Video On Demand* sont visés. L'interprétation peut changer, et en tout cas les termes de la licence changeront fin 2015, date de fin de la licence actuelle. Il semble justifié commercialement pour le MPEG-LA d'attendre cette date pour étendre la licence et inclure tout motif de distribution, surtout qu'à ce moment-là H.264 pourra être incontournable sur le Web : ce format représente déjà les deux tiers des vidéos distribuées sur le Web aujourd'hui.

Un nouvel espoir

Le second acte s'est joué en 2010 : Google rachète VP8, un codec de qualité équivalente à H.264, parfait pour la diffusion en *streaming*, puis le déclare gratuit. Dans la foulée, des accords sont conclus pour créer le format WebM, incluant Mozilla, la Free Software Foundation, des fabricants de matériel – le décodage matériel est vital pour que les codecs soient viables sur mobiles ou sur tout autre objet qu'un PC (téléviseur, box...) – et surtout Adobe².

La prophétie annonce que Flash, décidément taillé pour un rôle bipolaire, devrait également supporter WebM, tuant ainsi l'empereur H.264 (doucement) et sacrifiant probablement sa propre existence (doucement aussi). Malheureusement malgré les annonces en fanfare de mi-2010, ni Google ni Adobe n'ont vraiment tenu leurs promesses : Chrome supporte toujours nativement H.264, donc les éditeurs de site web n'ont pas de pression de côté-là, et Flash ne supporte toujours pas WebM, donc

1. Le résumé officiel de la licence : http://www.mpegla.com/main/programs/avc/Documents/AVC_TermsSummary.pdf.

2. Liste des membres du projet WebM : <http://www.webmproject.org/about/supporters/>.

les éditeurs de site web n'ont pas réellement d'incitation au changement. Sans carotte ni bâton, pourquoi changer ?

D'un autre côté le MPEG-LA fourbit ses armes juridiques prévenant qu'elle a des brevets tellement bien faits qu'elle peut attaquer quiconque fait un autre codec. Pour le suspense, notons que le consortium avait fait les mêmes menaces contre Theora sans jamais attaquer, et qu'il y a actuellement une enquête de la justice américaine concernant cette pratique de la menace. Bref cet épisode est encore à écrire, sera surtout juridique et on espère que le dénouement se fera avant 2016.

Les rebelles de l'open source se sont partiellement sacrifiés au passage, le niveau technique et la compatibilité du codec vidéo Theora n'étant pas suffisant par rapport aux exigences du marché. Par contre WebM inclut en plus de VP8 le codec audio Ogg Vorbis.

En bref : il y a de fortes chances que WebM soit le concurrent valable et gratuit que les sites attendaient, mais le doute juridique persiste.

6.3.2 Choisir son camp

En pratique, quelle stratégie adopter concernant l'encodage ? Regardez les tableaux 6.1 et 6.2 de compatibilité.

Tableau 6.1 — Table de compatibilité codec vidéo / navigateur.

H.264	VP8
Flash	
Internet Explorer 9	IE9*
	Firefox 4
Chrome	Chrome
Safari	
	Opéra
iOS	
	Opera Mobile**
Android**	Android**

* seulement si l'utilisateur a installé le codec sur sa machine

** dépend du mobile

Les grands absents sont bien sûr les navigateurs IE6, 7 et 8 ainsi que les anciennes versions des autres navigateurs (peu répandus). Ils sont cependant représentés par Flash, ce qui aujourd'hui fait largement pencher la balance en faveur de H.264 / MP3. Au moment d'écrire ces lignes, Flash a annoncé depuis deux ans le support de WebM (VP8 + Vorbis), mais cela n'est pas encore effectif.

Aujourd'hui il n'y a donc guère que deux stratégies (et un bonus) :

- Le stockage en double ne vous coûte rien : vous faites très peu de multimédia et payez votre stockage au forfait (sites vitrine, blogs...), le double encodage est parfait. Ou vous êtes trop riche.

Tableau 6.2 — Table de compatibilité codec audio / navigateur.

MP3	Vorbis
Flash	
Internet Explorer 9	
	Firefox 3.6+
Chrome	Chrome
Safari	
	Opéra
iOS	
	Opera Mobile
Android*	Android*

* dépend du mobile

- Vous devez assurer un maximum de compatibilité pour un coût minimal : c'est-à-dire un site classique qui hébergerait beaucoup de vidéos ou d'audio. Si pour supporter 2 formats cela vous coûte le double en stockage, alors restez sur H.264 / MP3 aujourd'hui mais préparez-vous à changer pour VP8 et Vorbis lorsque Flash les supportera.
- Vous vous appelez Youtube : vous devez aussi assurer une compatibilité maximale, mais il se trouve que votre maison mère possède 1 % des serveurs de la planète, et accessoirement le codec VP8. Dans ce cas-là effectivement le stockage massif de vidéos en deux formats ne vous fait pas peur et même se justifie.

En bref : H.264 c'est le présent, et WebM un futur aussi radieux qu'hypothétique. Flash est l'arbitre du jeu, et les sites n'auront un intérêt financier à changer que lorsqu'une version de Flash supportant WebM sera suffisamment répandue. Pour les *smartphones* certains commencent à sortir avec des puces de décodage WebM mais H.264 reste ultra-dominant, au point que sur cet environnement même Mozilla a dû sacrifier son engagement pour laisser une chance à Firefox Mobile de s'imposer.

6.3.3 Après le codec : les formats

Choix des formats

Ce livre n'a pas vocation à expliquer toutes les subtilités de l'encodage vidéo, qui est un métier en soi. Nous allons nous contenter de vous indiquer les formats et niveaux d'encodage les plus classiques des sites web, en partant des principes suivants :

- vous voulez de belles vidéos, mais aussi économiser votre bande passante et fournir une expérience fluide pour tous les utilisateurs ;
- vous utilisez la balise `video` lorsqu'elle est supportée, Flash lorsque ça n'est pas le cas ;

- vous ciblez aussi les mobiles, ce qui induit des bandes passantes et des écrans réduits.

Pour un support complet il vous faudra donc par ordre de priorité :

1. H.264 *high profile*, le son étant en MP3 ou en ACC. Ceci pour IE9, Safari et Flash avec un bon débit ou un bon processeur ;
2. H.264 *baseline profile*, en 640×480 ou en 480×360 pour les mobiles type iOS avec un débit 3G ;
3. WebM (VP8 et Vorbis), pour utilisation avec les balises natives dans Firefox, Chrome et Opéra, et pour prévoir le futur ;
4. 3GP en 320×240 ou en 177×144 pour les mobiles qui ne sont pas des *smartphones*.

Outils d'encodage

Pour un encodage manuel de quelques vidéos ou pistes sons, nous vous conseillons l'excellent Miro Video Converter¹ qui en plus d'être gratuit et open source est très simple d'utilisation et a préconfiguré plusieurs formats, y compris pour des mobiles comme iOS, Android et même certaines consoles.

Pour un encodage automatisé et plus industriel, la référence *open-source* est ffmpeg². Il est complet (ou complexe, comme vous le sentez) et est à ce titre embarqué préconfiguré dans bon nombre de logiciels client. Côté serveur vous pouvez le configurer pour encoder vos fichiers dans tous les formats que nous avons mentionnés, pour peu que vous respectiez la licence LGPL.

6.4 BON POUR LA PRODUCTION ?

Faisons le bilan de ce qu'on nous promet et de ce que nous avons aujourd'hui.

6.4.1 L'implémentation navigateur

Lecteur du futur, sache que cette partie n'est bien sûr valable qu'au moment d'écrire ces lignes, en avril 2012, et qu'il t'appartient de tester par toi-même les implémentations des navigateurs.

Comme on aime bien critiquer, nous allons surtout lister les défauts et voir que nous sommes encore en dessous de ce que propose un lecteur Flash.

1. <http://www.mirovideoconverter.com/>

2. <http://ffmpeg.org/>

Le plein écran : non mais oui

Peu d'implémentations du bouton de plein écran sont sorties : au moment d'écrire ces lignes (**avril 2012**), seuls Firefox, Safari et IE10 présentent un bouton d'accès au plein écran, interface pourtant indispensable pour l'utilisateur.

Les implémentations de lecteurs vidéo en production (Youtube étant le plus célèbre) pallient à ce manque de support natif en proposant un bouton « full viewport », qui consiste à donner simplement en CSS les dimensions de la page visible à l'élément **video**, puis à espérer que l'utilisateur pense à mettre son navigateur en plein écran...

Autre problème majeur : la spécification dit que vous ne devriez pas pouvoir déclencher un plein écran depuis JavaScript, pour des raisons de sécurité et pour ne pas embêter l'utilisateur. Autrement dit, c'est volontaire qu'une page ne puisse pas déclencher un plein écran sur l'élément vidéo. Heureusement, en plus du bouton natif qui commence à apparaître, tous les navigateurs récents se sont entendus sur une implémentation commune de la « Fullscreen API », capable de faire du plein écran sur n'importe quel élément. Voici donc comment implémenter un vrai plein écran sur un élément vidéo.

Bouton *fullscreen* pour Firefox et Webkit

```
<video id="mavideo" controls src="video.url">
</video>
<button id="fullscreen">FullScreen</button>
<style>
.fullviewport {
  width: 100%;
  height: 100%;
  position: absolute;
  background-color: black;
  left: 0;
  top: 0;
}
</style>
<script>
var oVideoDisplay = document.getElementById('mavideo');
document.getElementById('fullscreen').onclick = function(){
  // le standard
  if(oVideoDisplay.requestFullScreen) {
    oVideoDisplay.requestFullScreen();
  } // le préfixe Mozilla
  } else if(oVideoDisplay.mozEnterFullScreen) {
    oVideoDisplay.mozEnterFullScreen();
  } // webkit
  } else if(oVideoDisplay.webkitEnterFullScreen) {
    oVideoDisplay.webkitEnterFullScreen();
  } // fullscreen natif non supporté, passage en full viewport
  } else {
    oVideoDisplay.className = 'fullviewport';
  }
};
</script>
```



Tester en ligne !

<http://www.livre-html5.com/video/fullscreen.php>

Le code pour le *full viewport* marche partout, mais n'est bien sûr pas idéal, et il faudrait qu'il écoute les appuis sur la touche `Esc`. Le code de la FullScreen API est supporté par tous les navigateurs modernes et a le mérite de pouvoir s'appliquer sur n'importe quel élément d'une page : vous pourriez passer le conteneur de l'élément vidéo et de vos contrôles maison pour reproduire l'expérience d'un vrai lecteur vidéo, ou votre canvas pour un jeu vidéo ou bien encore n'importe quelle zone importante de votre site.

En bref : à l'instar de Flash le plein écran sur la vidéo est enfin possible, mais il faut penser à proposer ses propres contrôles.

Gérer le *streaming*

La seconde fonctionnalité majeure lors de la lecture des vidéos est la possibilité d'aller à un point précis du flux vidéo ou audio, même si celui-ci n'est pas encore téléchargé. Peu d'implémentations actuelles en sont capables (Webkit iOS) et certaines implémentations n'arrivent même pas à se déplacer dans une vidéo qui n'est pas entièrement chargée.

Ceci est dû à un manque de standardisation (volontaire) du protocole de *streaming* : le plus utilisé est RTMP qui est la propriété d'Adobe, mais HTTP est une technique viable. Au final, le choix du protocole dépend du serveur de *streaming*, et à ce jeu-là « Adobe Flash Media Server » est leader, puisque la spécification RTMP n'a été publiée que mi-2009, laissant peu de temps à un serveur *open-source* de sortir du lot.

En bref : c'est aux navigateurs d'essayer d'implémenter les protocoles répandus le plus rapidement possible, et cela n'est pas encore le cas.

Protection du contenu

Vous vous souvenez des sites qui interdisaient le clic droit sur les images pour « protéger le contenu » ? Le besoin et le fantasme du contrôle absolu sur tout contenu sont toujours aussi présents et est encore plus forts car il ne s'agit pas juste de photographes qui veulent protéger leurs droits mais de compagnies gérant des milliards de dollars.

Donc des plateformes de distribution de vidéo comme Youtube par exemple ne peuvent tout simplement pas se passer de Flash pour ces contenus, d'abord parce

qu'il implémente un protocole comme RTMPE (flux vidéo encrypté), et ensuite parce qu'il est réputé impossible de copier une vidéo distribuée *via* Flash. Ce dernier point est discutable : demandez au premier beau-frère venu la marche à suivre, il vous indiquera une bonne dizaine de *plugins* différents pour récupérer les vidéos sur Youtube, Dailymotion et autres grandes plateformes.

Le W3C n'a bien évidemment pas l'intention de spécifier un protocole qui limite la diffusion du contenu, car c'est tout simplement contraire à sa philosophie. Il y a peut-être une lueur d'espoir car, téléguidés par les intérêts de leurs maisons mères, certains navigateurs ont commencé à se regrouper entre eux pour s'accorder sur un tel format. Il est encore trop tôt pour savoir sur quoi ces discussions (fortement réprouvées par les libristes) finiront par aboutir.

En bref : si vous devez faire croire à vos ayants droit que leur contenu diffusé sera protégé (obligation de moyen), ne comptez pas à moyen voire à très long terme sur HTML5.

Des bugs à foison

Il semble décidément très dur de rattraper les années d'expérience de Flash en la matière : dès que vous sortirez du cadre de la lecture simple d'une piste vidéo ou audio, vous tomberez rapidement sur une multitude de bugs, en particulier sur les plateformes mobiles, qui sont d'autant plus pénibles à tester qu'elles sont nombreuses.

Des événements ne sont pas lancés, et ils le sont toujours de manière différente selon le navigateur, certaines actions comme le chargement *via* JavaScript d'un flux bloquent la lecture, des propriétés ne sont pas mises à jour... Il va vous falloir beaucoup d'expérimentations sur toutes les plateformes que vous visez (et ne comptez pas sur les émulateurs de mobiles pour reproduire fidèlement un bug) pour réussir à produire un code qui marche partout.

En bref : tant que vous utilisez les éléments multimédias dans le cas nominal, tout se passe bien. Dès que vous sortez de ce cas, vous tomberez sur un bug dans une plateforme ou une autre, ce qui est particulièrement gênant sur mobile où les navigateurs ne sont pas mis à jour fréquemment.

6.4.2 L'intégration avec la page

Au regard de la partie précédente sur l'implémentation navigateur, on comprend mal pourquoi on s'intéresserait à autre chose qu'à Flash. Pourtant les possibilités des balises multimédias, surtout vidéo, sont énormes lorsque l'on ajoute tout le reste des langages du Web comme CSS, Canvas, SVG ou WebGL.

Certains flasheurs ont probablement obtenu les mêmes résultats avec la seule techno Flash, mais si vous voulez éviter de vous disperser et réutiliser vos compétences web, ce sont les éléments natifs qu'il vous faut.

SVG

Honneur aux anciens. Présent depuis 10 ans, il n'y a que dans les navigateurs modernes (IE9 inclus) que SVG peut enfin s'exprimer. Les applications possibles vont de la décoration (bords arrondis, effet miroir...) aux boutons de l'interface en passant par des filtres sur la vidéo elle-même.

Passer une vidéo en noir et blanc

```
<video id="video" controls src="video.url"
  style="filter:url(#noir-et-blanc)">
</video>
<svg height="0">
  <defs>
    <filter id="noir-et-blanc">
      <feColorMatrix values="0.3 0.3 0.3 0 0
                          0.3 0.3 0.3 0 0
                          0.3 0.3 0.3 0 0
                          0 0 0 1 0"/>
    </filter>
  </defs>
</svg>
```



Voir la démo avec Firefox 4 !

<http://www.livre-html5.com/video/svg-filtre.php>

Ici on a défini un filtre noir et blanc en SVG avec la balise `filter`, et on l'a appliqué *via* CSS sur l'élément `video`.

Concernant les interfaces, il est possible avec SVG de mettre des formes, des transparences et des dégradés sur les éléments vidéo et de les intégrer naturellement à la navigation. Dans la capture d'écran suivante, le texte « Big Buck Bunny » est une vidéo.



Voir la démo avec Firefox

<http://svg-wow.org/video2/video.html>

En y rajoutant du JavaScript, on peut même coder des interfaces que l'on croyait réservées à Flash ou Silverlight. Dans une démo¹ de 2007 de Chris Double (développeur Mozilla), on peut voir l'intégration de la vidéo à l'intérieur d'éléments SVG (grâce à la balise `foreignObject`) qui peuvent être agrandis, rétrécies, et tournées par l'utilisateur, le tout avec des effets de transparence.

1. Démonstration utilisable avec Firefox 4 : http://double.co.nz/video_test/video.svg



Figure 6.3 — Une vidéo jouable comme élément décoratif.



Figure 6.4 — Interface SVG scriptée.

Notez que ces deux derniers exemples auraient pu être faits avec une technologie qui vous est sans doute plus familière : CSS3.

CSS(3)

Arrondis, dégradés, rotation, animations... cela ressemble furieusement à ce que nous promet CSS3 n'est-ce pas ? Effectivement le fait de se retrouver avec une vidéo dans un élément natif permet de la styliser aussi facilement que la première `div` venue. Non seulement le positionnement se fait naturellement (au contraire de Flash qui avait tendance à passer systématiquement par-dessus certains éléments), l'intégration dans le texte est naturelle et vous pouvez appliquer vos `padding`, `margin` et autre bordures normalement, mais avec CSS3 vous pouvez en plus rajouter quelques effets *bling bling* comme vous le feriez avec des images.

Le code suivant par exemple suffirait presque pour un effet de pêle-mêle (comme sur la figure 6.5) et montre une petite vidéo légèrement penchée, qui s'agrandit et se remet droite lorsque la souris passe au-dessus.

Agrandissement et rotation

```
<video id="mavideo" controls src="video.url"></video>
<style>
/* au repos, la vidéo est tournée et petite */
#mavideo {
  border: 5px solid black;
  width: 400px;
  background-color: white;
  transform: rotate(-45deg);
}
/* lorsque la souris arrive, on lui donne sa vraie largeur
* et on la remet horizontale
*/
#mavideo:hover {
  width: 890px;
  background-color: black;
  transform: rotate(0);
}
/* Pour le fun, on met une transition douce
* lors du passage entre les deux états
*/
#mavideo,
#mavideo:hover {
  transition: all 0.5s ease-in;
}
</style>
```



Voir la démo !

<http://www.livre-html5.com/video/css3.php>

Nous n'avons pas mis les préfixes constructeurs des propriétés CSS3 **transform** et **transition**, et IE9 ne gère pas l'animation en CSS. Qu'à cela ne tienne, sur ce navigateur nous voyons tout de même la rotation, l'agrandissement et le changement de fond.

Canvas

Canvas est le petit jeune qui monte de HTML5, car il permet une infinité d'effets visuels qui plaisent bien aux journalistes (oui, comme Flash il y a 10 ans et alors ?). Son usage principal est la manipulation de pixels en JavaScript. Vidéo... Pixels ... Il y a là de quoi faire des choses énormes comme des filtres, de l'analyse de vidéo et de la recomposition d'image.

Il est par exemple très facile de prendre un instantané d'une vidéo et de l'analyser.

Instantané d'une vidéo

```

<video id="ma-video" controls src="video.url"></video>
<canvas id="mon-canevas" width="850" height="480"></canvas>
<style>
#ma-video {
padding: 50px;
transition: background-color 0.7s linear;
}
</style>
<script>
var oVideo = document.getElementById('ma-video'),
oCanvas = document.getElementById('mon-canevas').getContext('2d'),
iWidth = 850,
iHeight = 480;
function makeAmbilight() {
// on recopie l'image actuelle dans le canevas
oCanvas.drawImage(oVideo, 0, 0, iWidth, iHeight);
// on peut obtenir un tableau des pixels de l'image
var aPixels = oCanvas.getImageData(0, 0, iWidth, iHeight);
// ...
// quelques opérations mathématiques plus tard
oVideo.style.backgroundColor = 'rgb(' + color + ')';
};
// on prend un instantané toutes les secondes
setInterval(makeAmbilight, 1000);
</script>

```

Ici nous avons donc une vidéo dont nous recopions l'image toutes les secondes dans un `canvas`, nous en extrayons les pixels et nous calculons ligne à ligne les valeurs Rouge Vert Bleu. Nous en faisons la moyenne et nous redéfinissons la couleur de fond de l'élément `video`. Pour que le fond se voie, on rajoute un `padding` CSS et pour le fun on rajoute une transition CSS3 !



Voir le code !

<http://www.livre-html5.com/video/canvas-ambilight.php>

Si vous voulez un code plus abouti, allez voir cette démo¹ où Canvas est utilisé pour la restitution de l'ambiance.

Des téléviseurs sur ce principe se vendent 1 000 €. Nous, on vous offre ce code parce qu'on est comme ça et que ça nous fait plaisir. Pour les frais de livraison, voyez avec votre fournisseur d'accès.

1. <http://media.chikuyonok.ru/ambilight/>



Figure 6.5 — Un effet *ambient* plus abouti.

À partir du moment où on a accès aux pixels d'une vidéo, des applications inimaginables sans plugins ou sans logiciel sont envisageables, comme des trucages vidéo avec fond vert, des effets spéciaux, de la reconnaissance faciale ou de la reconnaissance de mouvement. Il n'y a plus qu'à.

WebGL

L'autre spécification qui fait rêver, c'est la 3D nativement dans les pages. Il n'y a en fait aucun plugin qui jusqu'ici a réussi à être suffisamment solide pour afficher de la vraie 3D dans une page web. Flash, Java, ActiveX, tous avaient de sérieuses limites de performances. Le navigateur est extrêmement adapté à son environnement aussi les codeurs 3D auront accès à l'accélération graphique, ce qui leur manquait jusqu'ici.

La vidéo étant maintenant un élément comme les autres, elle peut être intégrée naturellement dans un environnement WebGL. Il serait difficile et inutile d'essayer de faire de meilleures démos que Mozilla, aussi nous vous conseillons d'aller voir leurs démos « 360° video »¹ et « the navigator »². Elles démontrent avec brio ce que l'on peut faire avec les technologies web et un peu de talent.

D'après les auteurs, il s'agit simplement de créer une boule en 3D, d'inclure une balise `video` pointant sur une vidéo spécialement préparée, de rajouter une minicouche de JavaScript pour gérer le déplacement de la souris et c'est tout...

1. <https://mozillademos.org/demos/immersivevideo/demo.html>

2. <https://mozillademos.org/demos/flight-of-the-navigator/demo.html>



Figure 6.6 — Démo. d'une vidéo dans un univers 3D.

En résumé

La moitié de vos visiteurs peuvent déjà avoir une expérience améliorée libre de tout plugin lorsqu'ils viennent voir vos vidéos. Pour de la diffusion simple de vidéos, nous vous conseillons clairement une implémentation contenant une interface à vous en HTML, et un mode de lecture *dual* HTML5 avec *fallback* en Flash. C'est d'ailleurs ce que proposent à peu près toutes les librairies vidéo JavaScript.

Vous n'êtes cependant pas prêt d'arrêter d'utiliser Flash sur bureau, et si vous avez des contraintes particulières de protection de contenu ou de diffusion en *streaming* cela restera probablement votre seul lecteur pendant longtemps. Concernant les codecs nous avons vu qu'il vaut mieux être attentiste et passer sur WebM lorsqu'une version de Flash le supportant sera suffisamment répandue.

Dans des environnements sans Flash ou bien maîtrisés (intranet, mobiles, consoles, TV ou démos techniques) ces deux nouveaux éléments sont d'ores et déjà indispensables. La parfaite intégration avec les autres technologies du Web promet des applications créatives.

7

Canvas

Objectif

Au cours de ce chapitre nous allons explorer l'immense monde qui s'offre à nous avec l'API Canvas. Autant vous prévenir tout de suite, il s'agit de l'une des API les plus conséquentes d'HTML5. Nous allons ici apprendre à créer et modifier du contenu graphique étape par étape, partant du simple tracé de rectangle jusqu'à la création de filtre à la Photoshop. Nous verrons également le fonctionnement des animations et des contrôles pour que vous ayez la base de connaissance minimale pour aborder le monde du jeu vidéo.

7.1 LES BASES

7.1.1 L'API Canvas 2D

Si l'on devait donner une définition stricte, nous pourrions dire que l'API Canvas est une API bas niveau de dessin. Le W3C ne définit pour le moment qu'une API 2D, qui est celle que nous allons utiliser au cours de ce chapitre. Elle permet de créer du contenu graphique 2D sur une surface, définie par la balise `canvas`.

Concrètement, nous allons pouvoir dessiner à peu près n'importe quoi dans cette zone rectangulaire, par exemple des droites, des courbes, des rectangles, des cercles, des images, du texte, etc., et tout ce qui y est dessiné est en permanence accessible en détail par un tableau de pixels.

L'utilisation de Canvas n'a pour limite que votre imagination : rognage de photo, analyse d'image ou de vidéo, application de filtres de modification de couleurs, animations, jeux vidéo... Bref, un peu de tout, tant que ça reste en 2D.

Note : niveau prononciation du mot *canvas*, on entend un peu de tout ! Les Anglais disent « kenvasse », les bons français utiliseront le mot canevas, et les Français habitués à en entendre parler en anglais finiront par dire naturellement « kannevasse », sans parler du fait que l'on puisse le prononcer « an » (comme un « banc ») ou « anne ». Il n'y a pas vraiment de bonne réponse, vous devrez donc choisir votre camp !

7.1.2 Mettre en place notre terrain de jeu

Tout d'abord, nous déclarons un *canvas* possédant un *id*, une largeur (*width*) et une hauteur (*height*), et rédigeons un petit message qui sera affiché en cas de manque de support (nous pourrions également utiliser une image ou n'importe quel autre élément HTML).

Déclaration du canvas

```
<canvas id="mycanvas" width="500" height="300">  
  Il faut vivre avec son temps mon vieux !  
</canvas>
```

Par défaut, un *canvas* est une zone blanche. Donc pour pouvoir voir quelque chose, nous ajoutons une petite bordure en CSS :

```
<style>  
  canvas{border:1px solid}  
</style>
```

Ce qui produira la zone visible sur la figure 7.1.



Figure 7.1 — Notre zone de jeu.

C'est donc dans ce petit espace que nous allons pouvoir commencer à dessiner. Mais avant ça, il faut récupérer le *contexte* du *canvas*. C'est en fait via ce contexte que l'on peut dessiner. On l'obtient de la manière suivante :

Récupérer le contexte du canvas

```
oCtx = document.getElementById("mycanvas").getContext("2d");
```

Le paramètre de `getContext()` correspond à l'identifiant du type de contexte que l'on souhaite utiliser. La spécification officielle du W3C définit le contexte `2d`, qui va donc être celui que nous allons utiliser puisque nous ne souhaitons pas faire de 3D.

Note : si nous avions voulu utiliser WebGL pour la 3D, nous aurions utilisé l'identifiant `experimental-webgl`.

Une fois le contexte obtenu nous utiliserons des fonctions indépendantes pour chaque dessin que nous allons réaliser.

Ainsi, pour plus de lisibilité, voici le code de base qui va appeler nos fonctions de dessin.

Structure de base de notre code au cours de ce chapitre

```
var oCtx = null;
window.addEventListener('load', function() {
  oCtx = document.getElementById("mycanvas").getContext("2d");
  dessinerUnTruc();
}, false);
function dessinerUnTruc(){
  // C'est ici que toute la magie opère !
};
```

Note : pour une fois, nous utiliserons des noms de fonctions en français afin de ne pas les confondre avec les méthodes de la Canvas API.

7.1.3 Le repère

Les coordonnées du point 0, 0 avec `canvas` correspondent au coin **supérieur gauche** de la surface de dessin. Si vous vous souvenez bien, notre balise `canvas` a une largeur de 500 pixels et une hauteur de 300 pixels.

Par conséquent, notre repère correspond à la figure 7.2.



Figure 7.2 — Le repère de notre `canvas`.

7.1.4 Dessiner des formes basiques

Tracer un simple rectangle

La méthode `fillRect()` du contexte permet de dessiner un rectangle en spécifiant en paramètres le décalage horizontal, le décalage vertical, la largeur et la hauteur.

Tracer un rectangle

```
function dessinerRectangle(){  
    oCtx.fillRect(100, // position X du coin supérieur gauche  
                  100, // position Y du coin supérieur gauche  
                  300, // largeur  
                  50); // hauteur  
};
```

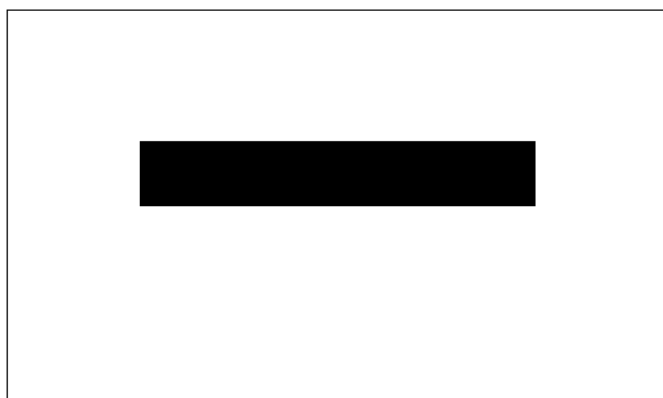


Figure 7.3 — Un bien beau rectangle !

Changer la couleur

Vous remarquerez que par défaut, la couleur utilisée est le noir. Si l'on veut changer cette couleur, il est nécessaire de l'indiquer au contexte avant de remplir la zone rectangulaire. Pour cela nous attribuons la valeur hexadécimale du gris à l'attribut `fillStyle`.

```
function dessinerRectangleGris(){
  oCtx.fillStyle = "#cccccc";
  oCtx.fillRect(100, 100, 300, 50);
};
```

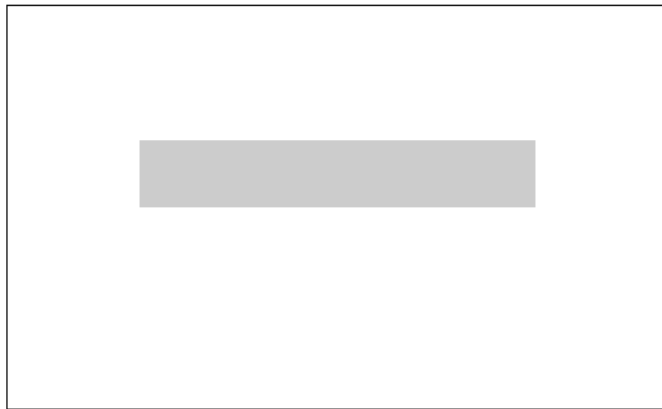


Figure 7.4 — Un superbe rectangle gris !

Tracer un trait

Le tracé d'un trait avec l'API Canvas s'effectue en plusieurs étapes :

- Le démarrage du tracé avec `beginPath()`.
- Le déplacement jusqu'au point du début du tracé avec `moveTo()`.
- La réalisation du tracé jusqu'à un second point avec `lineTo()`.
- Faire apparaître le tracé avec `stroke()`.

Tracer un trait

```
function dessinerTrait(){
  // on démarre le tracé d'une forme
  oCtx.beginPath();
  // on se place au bon endroit
  oCtx.moveTo(100, 100);
  // on trace une ligne jusqu'au point (200, 200)
  oCtx.lineTo(200, 200);
  // on fait apparaître cette ligne
  oCtx.stroke();
}
```

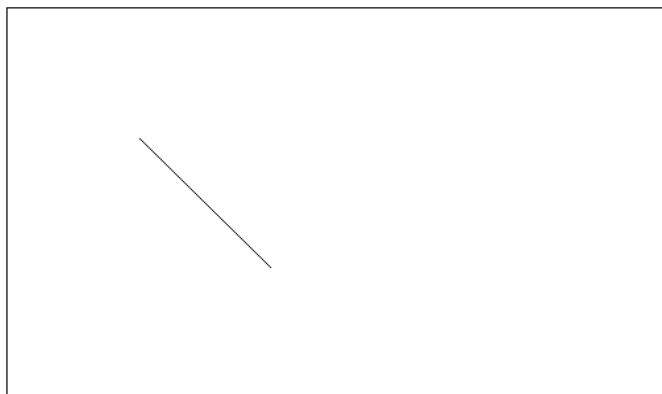


Figure 7.5 — Tracer un joli trait.

Ça semble plutôt compliqué pour pas-grand-chose n'est-ce pas ? Dans notre cas c'est sûr, car on réalise un seul trait, mais vous allez voir que ces méthodes sont plutôt pratiques pour créer des formes plus complexes.

Note : si l'on faisait une analogie par rapport au dessin papier, la méthode `moveTo()` correspondrait au levé du stylo pour se déplacer vers un point, et la méthode `lineTo()` au tracé.

Dessiner un triangle

Pour dessiner un triangle, rien de nouveau, on ne fait que continuer le tracé de la forme jusqu'à revenir au point de départ.

Dessiner un triangle

```
function dessinerTriangle(){
  oCtx.beginPath();
  oCtx.moveTo(100, 100);
  oCtx.lineTo(200, 200);
  oCtx.lineTo(200, 100);
  oCtx.lineTo(100, 100);
  oCtx.stroke();
}
```

Nous pourrions styliser un peu ce tracé en utilisant l'attribut `strokeStyle`, qui fonctionne exactement comme le `fillStyle` utilisé sur le rectangle un peu plus haut.

fill et stroke

Vous avez probablement remarqué que lorsque l'on dessinait notre carré nous utilisions plutôt des méthodes qui parlent de « fill », et qu'à présent les traits parlent plutôt de « stroke ». Vous rappelez-vous de Paint sur Windows ? Vous pouviez dessiner des rectangles en faisant apparaître : leur contour, leur contenu ou les deux.

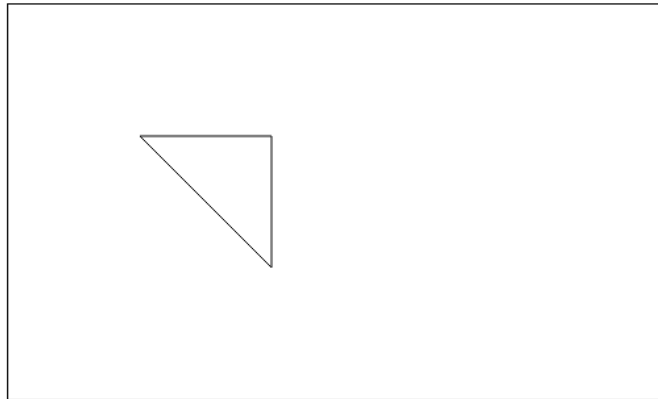


Figure 7.6 — Un beau petit triangle rectangle.



Figure 7.7 — Les différents types de tracé sous Paint.

Eh bien devinez quoi, ici c'est la même chose ! En fait, lorsque l'on trace un trait comme nous venons de le faire, nous traçons le contour d'une hypothétique forme. Ce contour s'appelle le *stroke*. Le contenu de notre forme, vous l'aurez deviné, c'est le *fill*. Par conséquent, si l'on reprend simplement l'exemple précédent de triangle, nous pouvons très facilement faire ressortir le contour du contenu avec quelques styles :

Styliser le contour et le contenu de notre triangle

```
function dessinerTriangle(){  
    oCtx.beginPath();  
    oCtx.moveTo(100, 100);  
    oCtx.lineTo(200, 200);
```

```
oCtx.lineTo(200, 100);
oCtx.lineTo(100, 100);
// on élargit l'épaisseur du trait
oCtx.lineWidth = 10;
// on rend le contenu gris clair
oCtx.fillStyle = "#dddddd";
// on affiche le contenu
oCtx.fill();
// on rend le contour gris foncé
oCtx.strokeStyle = "#222222";
// on affiche le contour
oCtx.stroke();
}
```

Note : nous avons utilisé `lineWidth` pour spécifier la largeur de notre ligne de contour afin que vous puissiez y voir clairement le contour.

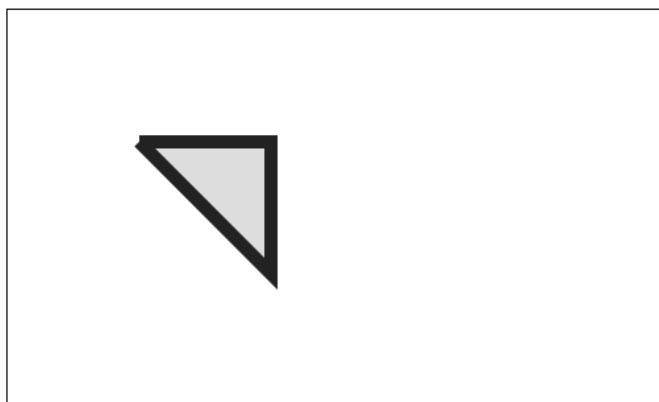


Figure 7.8 — Un sublime triangle avec un fantastique contour cassé.

Terminer le tracé

On voit ici clairement le fonctionnement du *stroke* et du *fill*. Seulement il y a un dernier petit souci, un des coins du triangle semble cassé. C'est tout à fait normal puisque l'API Canvas ne sait pas vraiment si nous souhaitons relier ces deux points ou non. Pour lui indiquer que c'est bien le cas et que notre figure est complète, nous devons utiliser `closePath()`, qui va donc logiquement de pair avec le `beginPath()` que nous avons ouvert au début, ce qui nous donne la figure 7.9.

Clôturer le tracé

```
function dessinerTriangleFerme(){
  oCtx.beginPath();
  oCtx.moveTo(100, 100);
  oCtx.lineTo(200, 200);
  oCtx.lineTo(200, 100);
```

```
oCtx.lineTo(100, 100);  
// fermeture du tracé  
oCtx.closePath();  
oCtx.lineWidth = 10;  
oCtx.fillStyle = "#dddddd";  
oCtx.fill();  
oCtx.strokeStyle = "#222222";  
oCtx.stroke();  
}
```

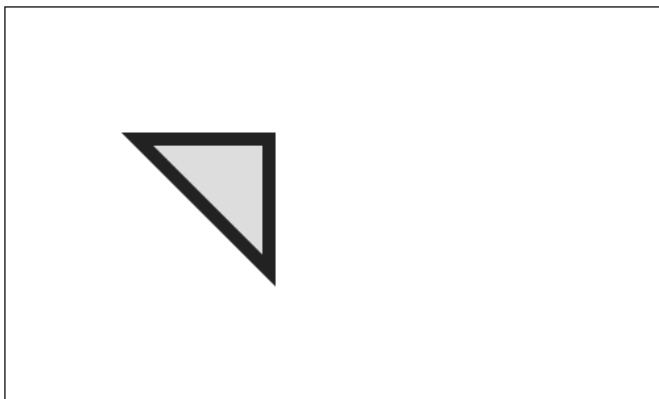


Figure 7.9 — La magnificence triangulaire à son paroxysme.

7.2 PASSONS AUX CHOSES SÉRIEUSES

7.2.1 Une bonne pratique : la translation de contexte

Avec la technique de tracé précédente, nous pouvons déjà tracer plein de jolies choses dans notre surface `canvas`. Cependant, cela va vite faire mal au crâne si on manipule en permanence de grandes valeurs de décalage horizontal et vertical qui oscillent entre 200 et 500 par exemple.

Une technique consiste à effectuer une translation du contexte jusqu'au lieu du dessin, ce qui permet de dessiner à partir du point (0, 0). Dessiner à partir de ce point présente plusieurs avantages. D'une part il améliore la compréhension du code car vous pouvez utiliser des valeurs négatives pour dessiner vers la gauche et vers le haut, et d'autre part vos dessins ne sont pas dépendants du lieu où ils sont tracés. Si vous décidez de déplacer votre dessin il suffira de modifier la translation plutôt que de modifier chacune des valeurs qui le constituent.

On peut comparer cette méthode avec le geste d'un dessinateur papier qui déplace sa main à différents endroits d'une feuille pour être plus à l'aise, plutôt que de galérer à atteindre la zone opposée de sa feuille à bout de doigts.

Pour pouvoir exploiter cette technique il faut tout d'abord sauvegarder la position initiale du contexte avec `save()`, la traduire avec `translate()`, effectuer le tracé comme nous savons déjà le faire, puis la restaurer avec `restore()`.

Effectuer une translation de contexte

```
function dessinerTriangleTranslation(){
  // sauvegarde de la position initiale du contexte
  oCtx.save();
  // translation du contexte
  oCtx.translate(250, 150);
  // les valeurs partent à présent du point (0, 0)
  oCtx.beginPath();
  oCtx.moveTo(40, 0);
  oCtx.lineTo(0, -60);
  oCtx.lineTo(-40, 0);
  oCtx.lineTo(40, 0);
  oCtx.closePath();
  oCtx.lineWidth = 10;
  oCtx.stroke();
  // restauration de la position initiale du contexte
  oCtx.restore();
}
```

Ainsi il est plus simple de se repérer, et le code est plus maintenable. La figure 7.10 correspond au tracé qui en résulte.

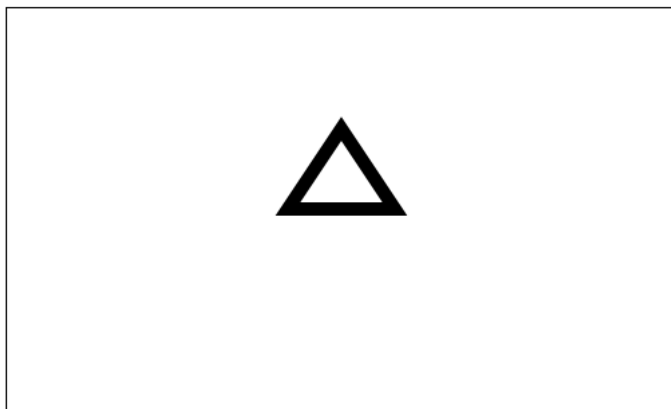


Figure 7.10 — Un triangle tracé avec translation de contexte.

7.2.2 Appliquer des ombres

La Canvas API met à disposition des méthodes pour générer des ombres, exactement comme le font les propriétés d'ombres du CSS3. Elles sont applicables sur du texte ou des formes. Tout comme en CSS3, il est possible de spécifier :

- Le décalage horizontal avec `shadowOffsetX`.
- Le décalage vertical avec `shadowOffsetY`.

- Le rayon de flou avec `shadowBlur`.
- La couleur de l'ombre avec `shadowColor` (en chaîne de caractères correspondant à du CSS).

Ainsi on peut enrichir l'exemple précédent avec :

Ombre notre triangle

```
function dessinerTriangleOmbre(){
  oCtx.save();
  oCtx.translate(250, 150);
  oCtx.beginPath();
  oCtx.moveTo(40, 0);
  oCtx.lineTo(0, -60);
  oCtx.lineTo(-40, 0);
  oCtx.lineTo(40, 0);
  oCtx.closePath();
  oCtx.lineWidth = 10;
  // le décalage horizontal
  oCtx.shadowOffsetX = 5;
  // le décalage vertical
  oCtx.shadowOffsetY = 5;
  // le rayon de flou
  oCtx.shadowBlur = 10;
  // la couleur
  oCtx.shadowColor = 'rgba(0, 0, 0, 0.3)';
  oCtx.stroke();
  oCtx.restore();
}
```

Note : l'ombre est appliquée sur tout le tracé actuel, que ce soit du contenu ou un contour. Ici on n'utilise que le contour du triangle (`stroke()`).

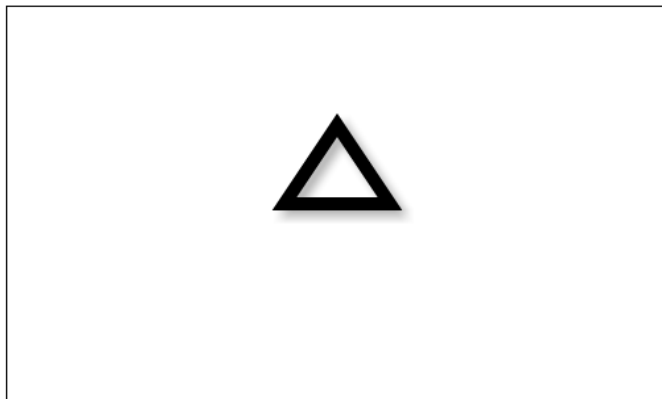


Figure 7.11 — Une légère ombre est appliquée sur le contour du triangle.

7.2.3 La Text API

Il est également possible d'écrire du texte dans un `canvas` ! On utilise pour cela la Text API, qui permet de styliser le texte de manière similaire au CSS.

Du texte est considéré comme une forme, exactement comme un rectangle. On peut par conséquent n'afficher que son contour avec `strokeText()` par exemple.

Il existe trois attributs utilisables pour paramétrer notre texte :

- `font`, qui correspond à la propriété du même nom en CSS, et qui permet de combiner de nombreuses informations (police, taille du texte, gras, italique, taille de la ligne...).
- `textAlign`, qui aligne le texte (peut prendre les valeurs `start`, `end`, `right` et `center`, et vaut par défaut `start`).
- `textBaseline`, qui permet de choisir la hauteur du pied de ligne de notre texte.

Nous allons transformer notre petit triangle ombré précédent en véritable logo en inscrivant le nom *Triangulous* en rouge juste en dessous. Nous ajoutons également une légère ombre sur le texte pour un rendu visuel plus doux.

```
function ecrireTitre(){
  oCtx.save();
  oCtx.translate(250, 200);
  oCtx.shadowOffsetY = 2;
  oCtx.shadowBlur = 2;
  oCtx.shadowColor = 'rgba(0, 0, 0, 0.3)';
  oCtx.fillStyle = "red";
  // on applique la propriété CSS souhaitée
  oCtx.font = "30px Calibri";
  // on centre le texte
  oCtx.textAlign = "center";
  // on l'affiche
  oCtx.fillText("Triangulous", // le texte à afficher
    0, // la position en X
    0, // la position en Y
    200); // la taille maximale de la zone de texte
  oCtx.restore();
}
```

7.2.4 Les dégradés

Les couleurs unies c'est bien gentil mais les dégradés ça fait plus pro. Et vous allez voir que ce n'est pas si compliqué à réaliser. Nous allons appliquer sur notre triangle un dégradé allant du blanc au noir, avec le noir en haut.

Pour cela on commence tout d'abord par créer le dégradé à partir du contexte et le stocker dans une variable grâce à la méthode `createLinearGradient()`.

```
var oGradient = oCtx.createLinearGradient(0, 0, 0, -60);
```

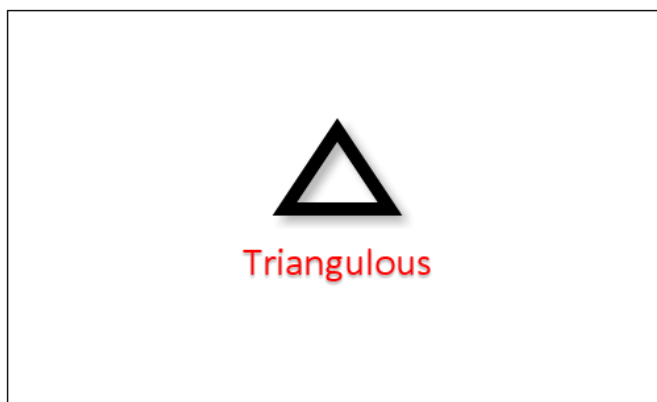


Figure 7.12 — Notre superbe logo de la marque « Triangulous »

Les deux premiers paramètres sont les coordonnées du point de départ du dégradé, et les deux derniers celles de sa fin. Dans notre cas au moment d'appliquer le style sur notre triangle nous avons effectué la translation de notre contexte, et partons donc du point (0, 0). Le bas du triangle est le point (0, 0) et le haut est situé en (0, -60).

Note : il existe également une méthode `createRadialGradient()` pour les dégradés radiaux.

Une fois le dégradé initialisé et placé, il suffit d'ajouter des points d'arrêts sur ce dégradé. Ces points ont un placement en pourcentage (situé entre 0 et 1) par rapport à la taille totale du dégradé, ainsi qu'une couleur. Il vous est possible de placer autant de points d'arrêt que vous le souhaitez selon la complexité de votre dégradé grâce à la méthode `addColorStop()` de votre objet de dégradé.

Exemple basique de placement de points d'arrêts de dégradé

```
oGradient.addColorStop(0, "white");  
oGradient.addColorStop(1, "black");
```

Dans notre cas nous allons plutôt faire démarrer le blanc à partir de la valeur 0.1, ce qui signifie que les 10 % précédents seront d'une couleur blanche unie. De cette manière nous allons cacher le côté bas du triangle (qui sera alors blanc sur blanc).

Enfin nous appliquons le dégradé sur l'attribut `strokeStyle` du triangle :

Attribuer le dégradé au style du contour

```
oCtx.strokeStyle = oGradient;
```

Code final du triangle avec dégradé

```
function dessinerTriangleOmbreDegradé(){  
  oCtx.save();  
  oCtx.translate(250, 150);
```

```
// initialisation du dégradé
var oGradient = oCtx.createLinearGradient(0, 0, 0, -60);
// déclaration des points d'arrêt des couleurs
oGradient.addColorStop(0.1, "white");
oGradient.addColorStop(1, "black");
oCtx.beginPath();
oCtx.moveTo(40, 0);
oCtx.lineTo(0, -60);
oCtx.lineTo(-40, 0);
oCtx.lineTo(40, 0);
oCtx.closePath();
oCtx.lineWidth = 10;
oCtx.shadowOffsetX = 5;
oCtx.shadowOffsetY = 5;
oCtx.shadowBlur = 10;
oCtx.shadowColor = 'rgba(0, 0, 0, 0.3)';
// attribution du dégradé au style du contour
oCtx.strokeStyle = oGradient;
oCtx.stroke();
oCtx.restore();
}
```



Figure 7.13 — Le même triangle avec un effet de dégradé.

Note : nous ne les abordons pas dans ce livre, mais sachez qu'il est également possible de tracer des cercles, des lignes courbées (courbes de Bézier par exemple), ou encore d'effectuer des transformations comme les rotations, homothéties et ou des translations.

7.3 ACCÉDER AUX PIXELS DU CANVAS

Maintenant que nous avons vu ce qu'il était possible de créer de toutes pièces à l'intérieur d'un `canvas`, nous allons plutôt nous intéresser à la modification de son

contenu. Pour cela nous allons tout d'abord charger une image dans notre zone de dessin puis lui appliquer plusieurs filtres.

7.3.1 Intégrer des images

Il existe deux manières d'intégrer une image.

En direct (similaire à la balise `img`)

Le contexte de l'API Canvas possède une méthode `drawImage()` qui permet tout simplement d'afficher une image aux coordonnées désirées. Il suffit alors de créer un objet `Image`, lui attribuer une source `src`, et passer cette image à `drawImage()`.

Attention, ne pas faire ça !

```
var oLogo = new Image();
oLogo.src = "logohtml5.png";
oCtx.drawImage(oLogo, 120, 20);
```

Seulement le problème c'est que charger une image prend du temps. Si vous cherchez à accéder à une image distante, `drawImage()` s'exécutera avant même que la requête HTTP qui s'occupe d'aller chercher l'image n'ait eu le temps de décoller ! Il vous faudra par conséquent attendre que celle-ci soit chargée avant de la dessiner avec `drawImage()`. L'événement `onload` de l'image nous permet de déclencher la suite des événements une fois l'image chargée.

La bonne méthode pour charger une image

```
function dessinerLogo(){
    var oLogo = new Image();
    oLogo.src = "logohtml5.png";
    // fonction lancée uniquement une fois l'image chargée
    oLogo.onload = function(){
        oCtx.drawImage(oLogo, 120, 20);
    };
}
```

De cette façon, nous obtenons une magnifique image dans un `canvas` (figure 7.14).

En fond (similaire au `background CSS`)

La seconde méthode est l'utilisation d'un *pattern* à appliquer sur une surface. Ce *pattern* permet de créer des mosaïques en répétant une image infiniment en horizontal, vertical, ou les deux. C'est donc exactement le fonctionnement de la propriété `background-image` en CSS.

Pour ce faire, nous utilisons la méthode `createPattern()`, qui prend en paramètre une image et le type de répétition du motif (`repeat`, `repeat-x`, `repeat-y` et `no-repeat`). Nous appliquons ensuite ce *pattern* au style du *fill* ou du *stroke* sur lequel nous souhaitons voir placer la mosaïque.



Figure 7.14 — Une image dans un `canvas`, du jamais vu !

Réaliser une mosaïque en répétant une image

```
function dessinerFond(){
    var oMiniLogo = new Image();
    oMiniLogo.src = "minilogohtml5.png";
    oMiniLogo.onload = function(){
        // création du motif répété de mosaïque
        oCtx.fillStyle = oCtx.createPattern(oMiniLogo, "repeat");
        oCtx.fillRect(0, 0, 500, 300);
    };
}
```



Figure 7.15 — Une mosaïque qui ne pique pas les yeux (et évite les nœuds).

Et voilà ! Nous sommes fin prêts à traiter ce contenu

7.3.2 Les mains dans le cambouis : le tableau de pixels

Pour cette petite expérience, nous allons partir de l'image du logo « chargée à la manière d'un `img` » (figure 7.14) et voir ce à quoi nous avons accès.

Pour récupérer le tableau de pixels d'une certaine zone du `canvas`, il nous faut utiliser la méthode `getImageData()` du contexte. Cette méthode a besoin encore une fois du décalage horizontal et vertical, ainsi que de la largeur et hauteur de la zone ciblée. Dans le cadre de cet exemple, nous allons tout simplement utiliser toute la surface du `canvas`.

Récupérer le tableau de pixels de toute la surface du canvas

```
var oImageData = oCtx.getImageData(0, 0,
                                     oCtx.canvas.width,
                                     oCtx.canvas.height);
```

Une fois l'`ImageData` obtenu, nous allons pouvoir accéder à trois de ses attributs :

- `height` : le nombre de lignes de pixels,
- `width` : le nombre de colonnes de pixels,
- `data` : le tableau des pixels.

Donc là on se jette immédiatement sur l'attribut `data` comme des voraces en s'imaginant que l'on pourra faire quelque chose du genre :

```
oImageData.data[200][100].red = 255 ;
```

...Et c'est une belle désillusion ! Tout d'abord, le tableau `data` est un tableau à **une** dimension, ce qui signifie en gros que tout est en ligne et à la suite, et en plus, un pixel est représenté par quatre entrées successives du tableau représentant ses valeurs RGBA. Voici donc à quoi ressemble notre tableau :

```
[r1, g1, b1, a1, r2, g2, b2, a2, r3, g3, b3, a3, ...]
```

Note : chaque entrée du tableau a une valeur située entre 0 et 255, y compris l'alpha, qui est pourtant habituellement en pourcentage.

Cela va donc être un bon casse-tête de manipuler tout ça, et il va falloir être très minutieux pour ne pas s'emmêler les pinceaux !

Note : sachez également qu'une sécurité a été mise en place afin qu'un `canvas` présent sur un site A et incluant une image provenant d'un site B ne puisse pas accéder au tableau de pixels de cette image. Utiliser `getImageData()` sur un tel `canvas` déclencherait alors une exception. Si vous utilisez cette méthode en local avec des images vous devriez d'ailleurs travailler exclusivement sur `http://localhost/` plutôt qu'en direct via `file:///` pour éviter de déclencher l'exception.

7.3.3 Création d'un filtre noir et blanc

Nous voilà donc face à un tableau de $500 * 300 * 4$ entrées. Commençons tout d'abord par créer une boucle générique de parcours de notre tableau pixel par pixel.

Parcourir les pixels un par un

```
for (var i = 0; i < oImageData.data.length; i = i + 4){  
    // faire quelque chose avec oImageData.data[i] (valeur rouge)  
    // faire quelque chose avec oImageData.data[i + 1] (valeur verte)  
    // faire quelque chose avec oImageData.data[i + 2] (valeur bleue)  
}
```

Très bien, on peut donc à présent travailler sur chaque pixel.

Pour appliquer un filtre de noir et blanc, nous n'avons à toucher qu'aux valeurs RGB, et pas à la transparence. C'est déjà ça de gagné.

La question est maintenant « *Comment transforme-t-on du RGB vers du noir et blanc ?* ». Une première idée serait de faire la moyenne des 3 couleurs, puis d'appliquer cette même moyenne aux trois pixels. Effectivement, ça marche, mais au détail près que les couleurs sont dans ce cas très contrastées et crues. En fait, lorsque les logiciels de retouche graphique convertissent une photo RGB vers du noir et blanc (ou « niveaux de gris »), ils appliquent la pondération suivante :

- Le rouge vaut 30 % du niveau de gris.
- Le vert vaut 59 % du niveau de gris.
- Le bleu vaut 11 % du niveau de gris.

Nous n'entrerons pas dans le pourquoi du comment de la théorie des couleurs, et nous contenterons de traduire cette règle dans notre boucle de traitement des pixels.

Le traitement de conversion du RGB vers des niveaux de gris

```
// on calcule le niveau de gris que doit prendre le pixel  
var iGray = (oImageData.data[i] * 0.3 +  
             oImageData.data[i + 1] * 0.59 +  
             oImageData.data[i + 2] * 0.11);  
// on l'attribue à chaque valeur de couleur RGB  
oImageData.data[i] = iGray;  
oImageData.data[i + 1] = iGray;  
oImageData.data[i + 2] = iGray;
```

Une fois la boucle passée, la dernière chose à faire est de réinjecter ce nouveau tableau de pixels à la place de l'ancienne dans le `canvas`. Pour cela nous utilisons `putImageData()`, qui prend en paramètres le tableau de pixels, et les coordonnées du point d'injection.

Réinjecter le nouveau tableau de pixels dans le contexte

```
oCtx.putImageData(oImageData, 0, 0);
```

Code final de notre filtre noir et blanc

```
function filtreNoirEtBlanc(){
    var oImageData = oCtx.getImageData(0, 0,
                                         oCtx.canvas.width,
                                         oCtx.canvas.height);
    for (var i = 0; i < oImageData.data.length; i = i + 4){
        var iGray = (oImageData.data[i] * 0.3 +
                     oImageData.data[i + 1] * 0.59 +
                     oImageData.data[i + 2] * 0.11);
        oImageData.data[i] = iGray;
        oImageData.data[i + 1] = iGray;
        oImageData.data[i + 2] = iGray;
    }
    oCtx.putImageData(oImageData, 0, 0);
}
```



Figure 7.16 — Le logo est maintenant en niveaux de gris.

Vous apprécierez l'ironie d'avoir un livre en couleur pour faire des démos noir et blancs, mais si vous n'êtes pas totalement convaincus, allez donc voir la démo.



Voir la démo !

<http://www.livre-html5.com/canvas/filtre-noir-blanc.php>

Note : n'oubliez pas que le filtre doit être appliqué après que la photo originale ait été chargée, c'est-à-dire juste après le `drawImage()` de la fonction de callback.

7.3.4 Mouais...

Les plus sceptiques d'entre vous se disent probablement « *C'est bien beau de faire des filtres, mais à part changer le ton des couleurs on ne peut rien faire...* ». Détrompez-vous ! Avec un peu d'algorithmie et d'huile de coude, vous pouvez recoder n'importe quel effet visuel que vous pourriez trouver dans Photoshop. Car il n'y a pas que les modifications de couleur, il y a également tout ce qui est, transformations, déformations, rognage, bruit, flous, effets mosaïques, etc. Alors certes, un effet de flou gaussien vous demandera un peu plus que six lignes de code comme notre filtre noir et blanc, mais en tout cas dites-vous bien qu'avec le tableau de pixels, vous avez une liberté totale sur la manipulation de l'image.

Et puis si vous vous ennuyez avec les images, vous pouvez toujours passer aux vidéos ! Allez donc voir au rayon Canvas du paragraphe 6.4.2 sur la vidéo.

7.3.5 Sauvegarder l'image

Vous êtes tellement fiers de votre tout nouveau filtre que vous aimeriez sauvegarder l'image contenue dans `canvas` ? C'est possible ! L'objet `Canvas` possède une méthode `toDataURL()` qui convertit le contenu de la zone en image en URL en base64. Il vous suffit de spécifier le format voulu, par exemple `image/png`, puis de rediriger l'utilisateur vers cette URL.

Exportation du contenu du canvas en PNG lors du clic sur « save »

```
document.getElementById("save").addEventListener('click', function() {  
    window.location = oCtx.canvas.toDataURL("image/png");  
}, false);
```

Ici nous l'affichons dans le navigateur, mais étant donné que ce n'est jamais que du texte, vous pouvez également l'envoyer au serveur via une XHR (requête AJAX) pour sauvegarder votre œuvre !

7.4 LES ANIMATIONS ET LES JEUX

Dans cette dernière partie nous allons effleurer du bout des doigts le fonctionnement d'une animation et d'un jeu dans lequel l'utilisateur se déplace au clavier. Sachez que le domaine du jeu vidéo est soumis à des problématiques de performances énormes et qu'il nécessite des formations spécifiques. Si vous vous intéressez à ce domaine, cette partie devrait cependant vous y sensibiliser et retenir votre attention.

7.4.1 La boucle d'animation

Une animation est en fait une boucle infinie qui exécute un script toutes les n millisecondes. La notion d'intervalle de temps entre les exécutions du script est importante car elle impacte sur les FPS (images par secondes) de l'animation. La première chose à faire est donc de créer notre boucle d'animation.

Déclencher la boucle d'animation

```
window.addEventListener('load', function() {
  oCtx = document.getElementById("mycanvas").getContext("2d");
  setInterval(mainLoop, 16);
}, false);
function mainLoop(){
  // Exécutée toutes les 16 millisecondes
}
```

La méthode `setInterval()` déclenche une fonction toutes les n millisecondes. Une animation fluide s'affiche en général à une vitesse moyenne d'environ 60 images / secondes. Une simple division de $1\,000/60$ nous donne le chiffre 16 et des poussières. Nous déclencherons donc notre boucle principale `mainLoop()` toutes les 16 millisecondes.

À chaque exécution de notre boucle principale, il faut effacer le contenu de l'image précédente afin de ne pas superposer les nouveaux dessins aux précédents. Pour cela nous utilisons `clearRect()` du contexte du `canvas`, en spécifiant le décalage horizontal, vertical, la largeur et la hauteur.

Effacer tout le contenu du canvas

```
oCtx.clearRect(0, 0, 500, 300);
```

Nous allons maintenant afficher un petit rectangle qui va se déplacer dans notre zone de dessin.

7.4.2 Déplacer un rectangle

Commençons par créer un objet `oRect`, qui contient les coordonnées de sa position actuelle :

```
var oRect = {x:0, y:100};
```

Pour avoir un premier exemple d'animation, il nous suffit de modifier notre boucle principale pour avancer le rectangle d'un pixel par exécution.

```
function mainLoop(){
  oCtx.clearRect(0, 0, 500, 300);
  oCtx.fillRect(oRect.x, oRect.y, 100, 60);
  oRect.x++;
}
```

Lorsqu'on lance notre page HTML, le petit bloc noir se déplace automatiquement de la gauche vers la droite, comme tente de l'illustrer la figure 7.17.



Figure 7.17 — Ce bloc se déplace en fonction du temps vers la droite du canvas (la trainée sur l'image représente son déplacement)



Tester l'animation !

<http://www.livre-html5.com/canvas/animation.php>

Nous ne contrôlons pas (encore !) le bloc, celui-ci n'est pas très joli, mais reconnaissez que c'est un comportement plutôt impressionnant dans une simple page web, et sans Flash.

Le code final de notre animation

```
var oCtx = null;
var oRect = {x:0, y:100};
window.addEventListener('load', function() {
    oCtx = document.getElementById("mycanvas").getContext("2d");
    setInterval(mainLoop, 16);
}, false);
function mainLoop(){
    // on efface l'affichage précédent
    oCtx.clearRect(0, 0, 500, 300);
    // on place le nouveau rectangle
    oCtx.fillRect(oRect.x, oRect.y, 100, 60);
    // on le fait avancer d'un pixel vers la droite
    oRect.x++;
}
```

Si vous souhaitez réaliser une animation sans aucun contrôle, vous avez déjà ici une première base totalement fonctionnelle. Libre à vous d'afficher ce que vous voulez à la place du bloc noir, et de le déplacer comme bon vous semble.

7.4.3 Ajouter les contrôles pour diriger notre bloc

Un peu frustré par le caractère incontrôlable de notre petit bloc ? La différence entre une animation et un jeu vidéo réside principalement dans les contrôles qui permettent au joueur d'interagir avec l'animation. Nous allons donc dans cette partie permettre au joueur de déplacer le bloc à l'aide des flèches directionnelles de son clavier.

L'objectif de cette partie est de remplacer la ligne suivante par quelque chose de plus intelligent :

```
oRect.x++;
```

Nous aimerions en fait tester à chaque itération la direction dans laquelle le joueur souhaite se déplacer. Nous créons donc un petit objet `oMove` qui va gérer les déplacements et remplaçons la ligne précédente par ceci :

```
if (oMove.down) oRect.y++;
if (oMove.up) oRect.y--;
if (oMove.left) oRect.x--;
if (oMove.right) oRect.x++;
```

Nous ne devons bien sûr pas oublier de déclarer cette variable en dehors de la boucle principale :

```
var oMove = {up:0, down:0, left:0, right:0};
```

Le seul travail qu'il nous reste à faire est à présent de modifier cet objet `oMove` en fonction des touches appuyées et relâchées.

Pour détecter l'appui sur une touche, il faut utiliser l'événement `keydown` et `keyup`, dont les *callbacks* ont pour paramètre un objet contenant un attribut `keyCode` qui correspond au code de la touche pressée. Les codes des touches gauche, haut, droite, bas, sont respectivement 37, 38, 39 et 40.

Modification de notre objet `oMove` en fonction des actions au clavier

```
// déclenché lorsque l'utilisateur presse une touche
window.addEventListener("keydown", function(e){
  // en fonction de la touche pressée on modifie oMove
  if (e.keyCode == 40) oMove.down = 1; // la flèche du bas est pressée
  if (e.keyCode == 38) oMove.up = 1; // la flèche du haut est pressée
  if (e.keyCode == 37) oMove.left = 1; // la flèche de gauche est pressée
  if (e.keyCode == 39) oMove.right = 1; // la flèche de droite est pressée
}, false);
// déclenché lorsque l'utilisateur relâche une touche
window.addEventListener("keyup", function(e){
  if (e.keyCode == 40) oMove.down = 0; // la flèche du bas est relâchée
  if (e.keyCode == 38) oMove.up = 0; // la flèche du haut est relâchée
  if (e.keyCode == 37) oMove.left = 0; // la flèche de gauche est relâchée
  if (e.keyCode == 39) oMove.right = 0; // la flèche de droite est relâchée
}, false);
```

Et c'est tout ! Le bloc est maintenant contrôlable avec les flèches.



Tester le jeu !

<http://www.livre-html5.com/canvas/controles.php>

Note : certains se demandent probablement pourquoi il a fallu passer par une variable intermédiaire pour gérer l'appui des touches. Tout simplement parce que `keydown` est basé sur la gestion du clavier de l'OS, c'est-à-dire que si la touche reste appuyée, il sera déclenché toutes les *n* millisecondes (comme lorsque vous laissez appuyer la touche d'une lettre dans un champ texte), ce que nous ne contrôlons pas. De plus, notre variable intermédiaire permet de gérer l'appui sur plusieurs touches en même temps, et donc d'aller en diagonale, ce qui aurait été impossible sans.

Vous souhaitez aller plus loin ? Voici quelques idées que vous pourriez déjà implémenter facilement :

- appliquer une texture de vaisseau à notre bloc,
- afficher des vaisseaux ennemis,
- afficher un projectile lors du clic sur la barre espace,
- gérer les collisions de ce projectile avec les vaisseaux ennemis,
- ajouter WebSocket pour rendre votre jeu multi-joueurs (patience, ça viendra au chapitre 11 !)

7.4.4 WebGL

L'API Canvas permet également de créer des graphismes 3D via une technologie appelée WebGL. Le standard expérimental WebGL est maintenu par le groupe Khronos, qui s'occupe également de l'OpenGL. WebGL est basé sur OpenGL ES 2.0, qui est un dérivé de l'OpenGL. Ce dérivé se veut plus simple et moins gourmand en mémoire, ce qui convient particulièrement au monde du Web et du mobile.

Bien qu'expérimental, WebGL permet déjà aux navigateurs sur lequel il est activé de réaliser des graphismes impressionnants comme l'illustre la figure 7.18. Il s'agit d'une scène d'aquarium en 3D utilisant des modèles 3D, des textures, des effets de lumière, et un jeu de caméra. Le tout s'exécute de manière fluide dans le navigateur.

Nous n'aborderons pas du tout WebGL dans ce chapitre, qui mérite largement un bouquin à lui tout seul.

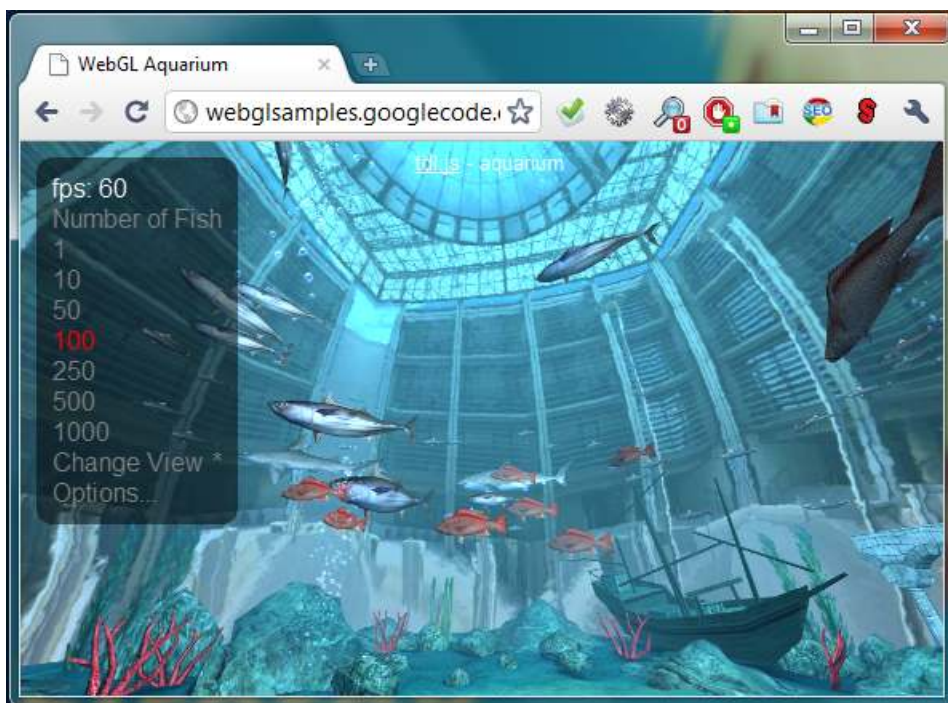


Figure 7.18 — L'aquarium WebGL.

<http://webglsamples.googlecode.com/hg/aquarium/aquarium.html>

Note : l'API Canvas se verra probablement un jour enrichi d'une spécification officielle concernant la 3D (donc rédigée par le W3C et non pas par Khronos), mais il y a fort à parier que WebGL soit la technologie 3D la plus répandue sur le web pendant un bon nombre d'années (voire qu'elle devienne à terme le standard, mais ça, seul l'avenir nous le dira !).

7.5 LES ALTERNATIVES

7.5.1 Pour Internet Explorer

C'est bien sympathique tout ça, mais ça marche sur quels navigateurs ?

L'API Canvas 2D que nous avons utilisé dans ce chapitre fonctionne sur tous les navigateurs sauf IE7 et 8. Ce qui signifie par conséquent que cela fonctionne sur IE9 ! C'est un bel effort de la part de Microsoft, qui comble petit à petit son retard.

Si vous souhaitez néanmoins faire fonctionner Canvas sur IE7 et 8, vous devriez vous intéresser à deux bibliothèques : ExplorerCanvas¹ (la plus répandue) qui fournira

1. <http://excanvas.sourceforge.net/>

une alternative VML supportée par Internet Explorer, et FlashCanvas¹, qui comme son nom l'indique, utilise Flash. Notez cependant que cette dernière est jusqu'à 30 fois plus performante, bien qu'elle demande d'acheter une licence si vous souhaitez l'utiliser pour un but lucratif.

Si jamais le navigateur ne supporte pas la balise `canvas`, alors tout comme pour les balises `audio` et `video`, c'est ce qui est contenu entre la balise d'ouverture et de fermeture qui est affiché :

Message affiché lorsque canvas n'est pas supporté

```
<canvas>
Ça ne doit pas être facile tous les jours d'avoir msn.com en page d'accueil.
</canvas>
```

7.5.2 SVG - Points forts et points faibles

Canvas n'est pas le seul standard qui permette d'afficher des éléments graphiques. SVG (*Scalable Vector Graphic*) en est un autre, et chacun présente des avantages et inconvénients. SVG est un format de données basé sur XML. Si vous utilisez du SVG au sein d'une page HTML, le DOM du SVG sera intégré à celui de la page HTML, qui bénéficiera alors d'une zone de dessin vectoriel.

Le redimensionnement

Par définition tout graphique SVG sera *Scalable*, c'est-à-dire redimensionnable sans perte de qualité. Vous pouvez zoomer infiniment sur une portion d'image vectorielle SVG sans jamais voir apparaître le moindre « effet escalier » de pixels.

Un Canvas, en revanche, ne saura faire que grossir l'image dessinée, laissant apparaître de gros pixels flous, exactement comme le ferait une image classique.

Les performances

SVG est globalement moins performant que Canvas, dont les performances sont constantes mais proportionnelles à la résolution de la zone de dessin. Pour les graphiques simples SVG conviendra très bien mais il perd en performances en fonction de la complexité du DOM. Cela le rend inutilisable lorsqu'il faut dessiner des scènes complexes. Canvas est donc beaucoup plus adapté pour les jeux par exemple.

Animations

Les animations sont natives et très faciles à réaliser en SVG. Ce n'est pas le cas de Canvas. Avec ce dernier, il vous faudra gérer vous-même la boucle d'animation comme nous l'avons fait au cours de ce chapitre.

1. <http://flashcanvas.net/>

Accéder aux images et aux vidéos

Canvas peut accéder aux pixels d'une image ou d'une vidéo. Ce n'est pas le cas de SVG.

Accessibilité

S'il y a bien une chose pour laquelle Canvas est vraiment mauvais, c'est l'accessibilité. Une zone de dessin Canvas est une boîte noire impénétrable de laquelle il est impossible de récupérer des informations pour les utilisateurs présentant des handicaps. Même la Text API qui permet d'écrire sur une surface Canvas n'est pas accessible aux lecteurs d'écran. Un texte écrit dans Canvas n'est d'ailleurs pas sélectionnable du tout, il s'agit uniquement de pixels, exactement comme un texte dans une image. SVG est quant à lui plus accessible, puisque son DOM est disponible.

Exportation

Canvas présente l'avantage par rapport à SVG de pouvoir exporter son contenu vers un fichier image, ce qui a été vu au cours de ce chapitre.

L'interactivité

Niveau interactivité, la balle est clairement dans le camp de SVG. Puisque celui-ci crée un DOM, chaque élément de ce DOM peut réagir aux événements (comme le survol de la souris). Par exemple, il est très simple de générer un graphique qui réagit lorsque l'on clique dessus pour afficher des informations détaillées.

Canvas est clairement très mauvais pour ça. La zone de dessin est totalement déconnectée du DOM et il est par défaut impossible de détecter sur quel élément passe la souris. Il vous faudra donc gérer absolument toutes les interactions de la souris à la main.

Pour détecter le clic sur un simple rectangle, vous devrez en JavaScript récupérer les coordonnées de la souris, puis tester si ces coordonnées sont situées dans le rectangle (et encore faut-il que vous ayez stocké les coordonnées de ce rectangle quelque part...), plutôt pénible donc.

Quand utiliser quoi ?

Pour résumer, voici les cas où chacun convient le mieux.

Canvas est plus adapté pour :

- générer des images bitmap (tableaux de pixels),
- éditer des images bitmap,
- analyser des images bitmap,
- réaliser des animations et des jeux complexes.

SVG est plus adapté pour :

- créer des interfaces indépendantes de la résolution,

- animer des éléments d'interface ou des composants interactifs,
- créer des graphiques et courbes interactives,
- éditer des images vectorielles.

Note : gardez cependant bien en tête que SVG et Canvas ne doivent être utilisés que lorsqu'il n'existe aucun équivalent en HTML et CSS. Ils ajoutent une couche de complexité supplémentaire, consomment en ressources, et réduisent l'accessibilité. À utiliser avec modération donc !

En résumé

La balise `canvas` cache derrière elle l'immense Canvas API, qui offre des possibilités infinies en termes de création et de modifications d'image. De nombreuses démonstrations très impressionnantes fleurissent sur le net, et son utilisation devient de plus en plus poussée. Reconnaissance faciale, analyse et modification vidéo en direct, jeux en 3D... Et tout cela dans le navigateur ! Elle est clairement l'une des principales révolutions d'HTML5, bien qu'elle ne soit pas simple à appréhender. Si vous souhaitez approfondir vos recherches à son sujet dirigez-vous vers des ouvrages spécialisés.

8

Envoyer des fichiers en glisser-déposer

Objectif

Dans ce chapitre nous allons particulièrement nous intéresser à l'expérience utilisateur et proposer un *uploader* de fichiers avec barre de progression et *drop* de fichiers depuis le bureau.

Cela nous permettra d'étudier plusieurs API : Drag and Drop, XHR2, la sélection multiple, la File API et la balise [progress](#).

8.1 LES ANCIENS FORMULAIRES D'ENVOI

Vous voilà de retour à la maison après quelques semaines de vacances au soleil au cours desquelles plusieurs centaines de photos ont été prises. Afin de faire râler les (futurs ex) amis, vous allez sur votre album photo en ligne préféré et il est probable que vous vous retrouviez nez à nez avec un bon vieux formulaire HTML4.

Outre l'esthétique (et encore ici on a de jolis arrondis sur les boutons, merci Apple), vous devez sélectionner vos photos une à une, avec ici un maximum de 6 par envoi, après quoi vous devrez attendre la fin d'un envoi pour recommencer. Autant dire que vous en avez pour des jours.

Figure 8.1 — Formulaire HTML d’envoi multiple classique

Ce qui serait bien plus ergonomique et efficace ce serait de pouvoir sélectionner vos centaines de photos dans l’explorateur de fichiers, et **glisser-déposer le tout dans le navigateur**, exactement de la même manière qu’on glisse-dépose une sélection de plusieurs chansons pour la mettre dans la *playlist* d’un lecteur audio.

Ce qui serait encore mieux, c’est que les fichiers soient **envoyés au serveur au fur et à mesure qu’on les dépose dans le navigateur**. En effet, on ne sait pas forcément avec exactitude quelles photos sélectionner parmi les 100 (il y en a bien quelques-unes de floues !), donc il peut être utile de pouvoir les ajouter en plusieurs fois.

Enfin, cerise sur le gâteau, on aimerait bien connaître l’avancement de chaque envoi par une belle **barre de progression**, plutôt qu’un *gif* animé qui tourne pendant quatre minutes sans que l’on sache où l’on en est.

La bonne nouvelle, c’est que HTML5 va permettre de remplacer les formulaires HTML4, grâce à la *Drag and Drop API*, la sélection multiple de fichiers et à la balise *progress*. Pour le transport nous utiliserons *XMLHttpRequest Level 2* et la *File API*.

8.2 LES FORCES EN PRÉSENCE

Un *uploader* multiple de fichiers, surtout avec glisser-déposer est un petit mille-feuille d’API, qui ne sont d’ailleurs pas toutes formellement classées dans HTML5. Nous allons d’abord présenter les bases dans cet ordre : point d’entrée (glisser-déposer et sélection multiple de fichiers), accès aux fichiers (*File API*) et suivi de l’envoi (API *XHR2* et balise *progress*).

8.2.1 « Et avec ça ? » ou la sélection multiple

Si vous disposiez déjà d’un formulaire d’envoi avec plusieurs sélecteurs de fichiers il ressemblerait probablement à ceci :

Notez l’utilisation des crochets dans le nom du champ

```
<form >
  <input type="file" name="user_file[]" />
  <input type="file" name="user_file[]" />
```

```



```

Ces multiples sélecteurs uniques seront remplacés par le sélecteur multiple unique (vous suivez ?)

Pour faire évoluer très simplement ces formulaires d'envoi, il suffit de rajouter l'attribut `multiple` au sélecteur de fichier. En plus, pour aider l'utilisateur nous rajoutons un filtre sur le type MIME des fichiers à sélectionner avec le nouvel attribut `accept`.

Exemple de sélecteur natif de médias, multiple

```
<input name="user_file" type="file" multiple accept="image/*, video/*" />
```

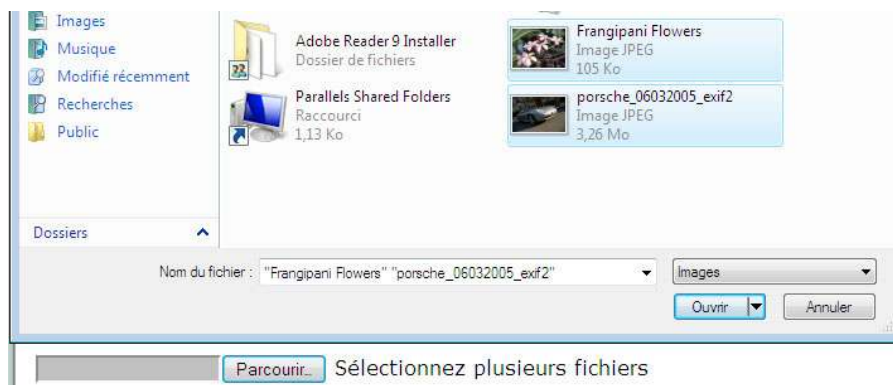


Figure 8.2 — Résultat sur Windows Seven.

C'est côté serveur que la modification la plus grosse doit être faite, si votre code était organisé pour ne recevoir qu'un seul fichier par envoi. Voici un changement qui se ferait en PHP.

Code initial : réception fichier unique

```

<?php
// affichage du nom du fichier envoyé
print $_FILES['user_file']['name'];
// affichage de son emplacement temporaire
print $_FILES['user_file']['tmp_name'];

```

La logique du code n'a ici prévu qu'un seul fichier par formulaire, la modification côté serveur consistera donc simplement à boucler sur la liste des fichiers pour répéter les règles pour chaque fichier.

Code final : réception fichiers multiples

```
<?php
// ajout de la boucle sur un sous-tableau (recommandation PHP.net)
foreach($_FILES['user_file']['error'] as $key => $error) {
    print $_FILES['user_file']['name'][$key];
    print $_FILES['user_file']['tmp_name'][$key];
}
```

PHP n'est pas très intuitif quant à l'accès aux fichiers multiples, mais qu'importe le flacon, pourvu qu'on ait l'ivresse d'accéder aux fichiers reçus.

Remarque : si vous vouliez simplement enrichir vos formulaires, sans barre de progression ni glisser-déposer de fichier, ce code serait déjà suffisant. Attention cependant car les fichiers sont envoyés en une seule requête et leurs poids s'additionnent, dépassant d'autant plus facilement les limites en place côté serveur. L'utilisateur tout heureux d'éviter la corvée de sélectionner ses fichiers un à un se retrouvera alors frustré et obligé d'additionner mentalement le poids des fichiers pour passer sous votre limite. Si vous avez besoin d'un uploader un peu plus « *pro* », passons aux API

8.2.2 Glissez, déposez (soufflez)

Au détour d'une conversation mondaine, cela fait toujours son petit effet que d'apprendre à vos convives que *IE 5.0* est le plus vieux navigateur supportant HTML5. L'API *Drag and drop* est en fait la formalisation de l'implémentation qu'en avait faite Microsoft à l'époque, bugs compris. En plus d'être imposante les développeurs web et même Ian Hickson (l'éditeur de HTML5) la jugent très alambiquée. Nous allons nous contenter ici du code nécessaire pour détecter qu'un objet vient d'être lâché dans notre page.

D'abord, que se passe-t-il lorsque l'on jette une image par exemple sur une page qui n'a pas prévu cette action ?

Balise nue sur fond blanc

```
<div> Rien de prévu </div>
```

Lorsque l'utilisateur tient l'image au-dessus de la page, le curseur lui indique qu'une action est possible. Ce curseur varie selon le système d'exploitation et le thème et correspond à ce que l'utilisateur a l'habitude de voir, ce qui est une bonne chose.

Il y a des variations selon le système :

- petite flèche de raccourci sur Windows XP et Vista (pas le même look) ;
- petite flèche avec texte « le déplacement » sous Seven ;
- gros signe « Plus » sur Mac OSX.

Lorsque l'utilisateur lâche l'image, elle s'affiche en entier. Le navigateur a pris la main et fait son boulot d'afficheur, ce qui explique que le curseur indiquait une action possible alors que nous n'avons encore rien écrit comme code.

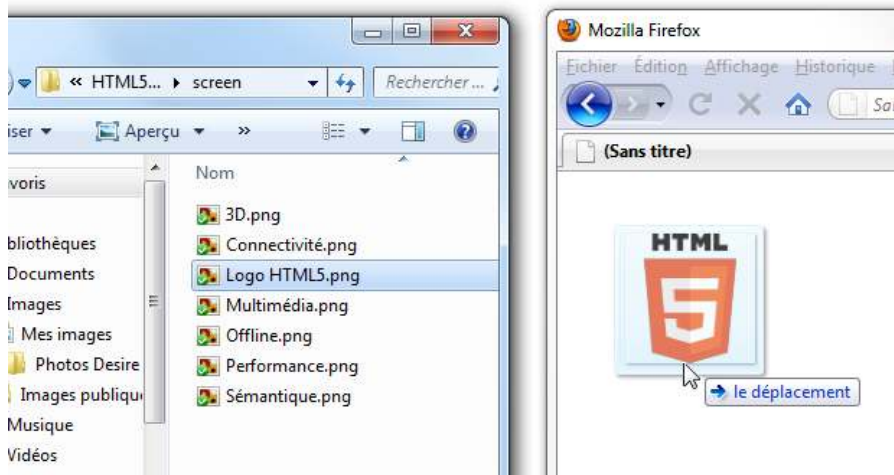


Figure 8.3 — Exemple de curseur de déplacement sous Windows Seven.

Au passage, du point de vue de l'ergonomie ce comportement ne nous arrange pas car nous voulons faire comprendre à l'utilisateur qu'il ne peut pas jeter son fichier n'importe où, mais à un seul endroit dans la page. Reportez-vous à la section complémentaire *Gérer le curseur pendant le drop de fichiers*.

Commençons déjà par voir comment :

1. définir une zone de réception,
2. détecter qu'un fichier a été jeté,
3. obtenir un pointeur vers ce fichier.

Récupération des fichiers jetés

```
<div id="drop-zone"> Largage ! </div>
<script>
// pointeur vers notre DIV cobaye
var oDropArea = document.getElementById('drop-zone');
// on écoute l'évènement 'dragover', exécuté très souvent
// tant que l'utilisateur est au-dessus de la zone
oDropArea.ondragover = function(e) {
    console.log('droparea.ondragover');
    // il FAUT annuler cet événement pour que le drop marche
    e.preventDefault();
};
// l'évènement qui nous intéresse : le 'drop'
oDropArea.ondrop = function( e ) {
    console.log('droparea.ondrop');
    // annulation du comportement du navigateur, qui veut afficher le fichier
    e.preventDefault();
    oDropArea.innerHTML = 'Fichier jeté et détecté';
    // à partir de là, notre code métier
};
```



Voir la démo !

<http://www.livre-html5.com/drop/drop.php>

Vous trouvez qu'il n'est pas intuitif d'annuler un événement (`.preventDefault()` dans `dragover`) pour qu'un autre (`drop`) puisse se déclencher normalement ? C'est à peu près ce que pensent tous les développeurs web qui sont passés par là avant vous mais il n'y a pas d'autre manière de faire. Chose amusante, le code que nous avons écrit jusqu'ici marche tout aussi bien sur IE6 que sur les navigateurs les plus récents, et la petite phrase de détection de l'action de drop s'affiche correctement. Voyons plutôt comment récupérer les fichiers ainsi jetés dans le prochain point sur l'API *File*.

8.2.3 L'API File

L'API *File* définit plusieurs interfaces qui permettent d'accéder aux données brutes (`FileReader`, `Blob`), de gérer les erreurs (`FileError`, `FileException`) ou d'accéder à un fichier sans risque pour l'utilisateur (*URI Scheme*). En sus il y a deux interfaces qui nous intéressent ici :

- `File` qui est un pointeur sur le fichier et contient ses métadonnées (poids, nom, type mime).
- `FileList` qui n'est qu'une collection d'objets `File`, mais c'est celle à laquelle les API donnent accès.

Cette API ne fait bizarrement pas partie de HTML5 (version W3C) et pourtant le seul moyen d'obtenir un objet `File` est de passer par une des API HTML5 comme *Drag and Drop* et notre `input type=file`. Pour des raisons de sécurité, le développeur ne pourra jamais instancier lui-même un objet `File`.

Récupération de la liste des fichiers sélectionnés

```
<form>
  <input name="user_file" id="user_file"
    type="file" multiple accept="image/*, video/*" />
  <input type="submit" value="Envoyez, Simone" />
</form>
<div id="afficheur">
  Fichiers sélectionnés : <span id="nb_files">0</span>
  <ul id="list_files"></ul>
</div>
<script>
// quelques références à nos objets DOM
var oInput = document.getElementById('user_file'),
    oStatNbFiles = document.getElementById('nb_files'),
    oListFiles = document.getElementById('list_files');
var iNbFiles = 0;
```

```
// À chaque nouvelle sélection, on met à jour la liste des fichiers


```



Voir la démo !

<http://www.livre-html5.com/drop/select-multiple-file-api.php>

Vous aurez remarqué que la référence à la liste de fichiers se trouve directement dans l'objet DOM du sélecteur, accessible par `.files`.

Voyons maintenant comment accéder à la même liste, mais après que l'utilisateur ait lâché ses fichiers sur notre page. On rajoute au code du *drag and drop* du chapitre précédent quelques lignes dans la fonction appelée par l'évènement `drop` :

Récupération des fichiers déposés

```
// l'évènement qui nous intéresse : le 'drop'
oDropArea.ondrop = function( e ) {
  // annulation du comportement du navigateur, qui veut afficher le fichier
  e.preventDefault();
  oDropArea.innerHTML = 'Fichier jeté et détecté';
  // à partir de là, notre code métier
  // on détecte si le navigateur sait accéder aux fichiers
  if('files' in e.dataTransfer) {
    var oLine;
    // On boucle sur la collection FileList pour récupérer nom et taille
    for(var i = 0; i < e.dataTransfer.files.length; i++) {
      oLine = document.createElement('LI');
      oLine.innerHTML =
        e.dataTransfer.files[i].name +
        ' ('+e.dataTransfer.files[i].size+' octets)';
      oListFiles.appendChild(oLine);
    }
  } else {
    // ce navigateur ne gère pas le glisser / déposer de fichiers
    oDropArea.innerHTML = 'Navigateur incapable';
  }
}
```

```

        oStatNbFiles.innerHTML = 'Un certain nombre';
    }
};

```



Voir la démo !
<http://www.livre-html5.com/drop/file-api.php>

Ici la liste des fichiers se trouve dans une sous-propriété de l'évènement `drop` : `drop.dataTransfer.files`. Dans un cas comme dans l'autre on a obtenu une liste d'objets `File` qu'il ne nous reste plus qu'à envoyer au serveur, avec la dernière couche de notre mille-feuille.

8.2.4 AJAX 2, le retour

Pour rester bons amis, évitons d'utiliser le terme AJAX, largement abusé, et parlons précisément de *XMLHttpRequest Level 2* qui est le nom de la spécification. Ou de XHR2 si vous êtes intimes (et nous allons l'être).

XHR2 permet au moins deux choses qui nous intéressent ici :

1. construire dynamiquement un formulaire et y inclure des pointeurs vers des fichiers, grâce à l'API associée `FormData` ;
2. surveiller la progression de l'envoi avec l'évènement `upload.onprogress`.

Créons de quoi sélectionner et envoyer au serveur un fichier, tout en surveillant la progression de l'envoi.

Envoi d'un fichier unique via XHR2

```

<input name="user_file" id="user_file"
    type="file" accept="image/*, video/*" />
<script>
var oXHR = new XMLHttpRequest();
document.getElementById('user_file').onchange = function() {
    oXHR.open('POST', '/upload.php');
    // la classe FormData se trouve dans le scope global
    var oFormData = new FormData( );
    // on recrée le formulaire
    oFormData.append(
        this.name, // récupération du nom du champ
        this.files[0] // pointeur sur le fichier sélectionné
    );
    // XHR2.send accepte maintenant des arguments, dont l'objet FormData
    oXHR.send(oFormData);
}
// un nouvel évènement pour suivre la progression de l'envoi
oXHR.upload.onprogress = function(e) {

```

```
var iPercent = (e.loaded / e.total) * 100;
console.log('fichier envoyé à '+iPercent+'%');
};
// l'évènement onload reste valable
oXHR.onload = function(e) {
    console.log(e.total+' octets ont été gaspillés pour cette démo');
};
</script>
```



Voir la démo !

<http://www.livre-html5.com/drop/select-multiple-file-api-xhr2.php>

Incroyable comme le code peut être simple lorsqu'on ne gère qu'un seul fichier et qu'on ne s'occupe ni de l'interface ni des erreurs ni de la compatibilité n'est-ce pas ? Pour le moment ce code ne fait que recréer un formulaire qui existait déjà et qui marcherait sans JavaScript... sauf que nous récupérons au passage le **pourcentage d'octets envoyés**, et nous allons voir tout de suite comment exploiter cette information.

8.2.5 Ça « progress » ?

Afin d'épargner à notre utilisateur d'avoir à surveiller les diodes réseau de sa(son) box/modem/routeur pour savoir si son fichier est bien en cours d'envoi, nous voudrions lui confirmer d'abord qu'on s'occupe de tout, que son fichier est bien en transit et qu'ensuite ça prendra un certain temps, que même lui pourrait deviner. Une **barre de progression** en somme. Miracle du HTML5, un élément a été prévu pour cela, il s'agit de la balise `progress`, qui prend les attributs `value` et `max` et qu'il suffit de modifier pour que le navigateur mette à jour une interface native.



Figure 8.4 — Barres de progression natives sur Windows.

Complétons le code du chapitre précédent en :

- rajoutant l'élément au HTML,
- mettant à jour les valeurs de `max` et `value` dans l'évènement `upload.onprogress`.

Détection du support natif et mise à jour de l'élément

```

<progress value="0" id="upload_progress"> fichier envoyé à 0 % </progress>
<script>
var oProgress = document.getElementById('upload_progress');
if('max' in oProgress) { // détection support natif
  oXHR.upload.onprogress = function(e) {
    oProgress.max = e.total;
    oProgress.value = e.loaded;
  };
} else { // méthode traditionnelle
  oXHR.upload.onprogress = function(e) {
    var iPercent = (e.loaded / e.total) * 100;
    oProgress.innerHTML = 'fichier envoyé à '+iPercent+'%';
  };
}
</script>

```



Vous pouvez voir une implémentation simple de barre de progression « à l'ancienne » à cette adresse : <http://www.livre-html5.com/balises/progress-meter.php>

Les navigateurs qui ont une interface pour `progress` ne vont pas afficher l'intérieur du contenu de la balise. Ce contenu HTML vous servira donc à montrer aux utilisateurs des autres navigateurs une simple phrase ou une barre de progression à coder vous-même.

Attention : ne pas confondre avec l'élément `meter` qui représente, lui, une jauge plutôt statique, et non pas une progression en cours. Les systèmes Windows et Mac OSX rendent bien compte de cette différence en rajoutant une animation visuelle sur l'élément `progress` tandis que `meter` ne bouge pas. `meter` peut par exemple servir à afficher un quota d'utilisation d'espace disque.

8.3 ASSEMBLAGE FINAL

Nous avons vu le détail pour chaque API, il y a un moment où il faut lier tout ça ensemble. Commençons par les points d'entrée.

8.3.1 Savoir recevoir

Formulaire, zone de drop, listing des fichiers et références JS

```

<form method="POST" target="/upload.php">
  <input id="user_file" type="file"
    multiple accept="image/*,video/*" />

```

```

    OU
  </form>
  <div id="drop-area">
    Posez vos fichiers ici
  </div>
  Fichiers sélectionnés : <span id="nb_files">0</span>
  <ul id="list_files"></ul>
  <script>
  // quelques références à nos objets DOM
  var oInput = document.getElementById('user_file'),
      oDropArea = document.getElementById('drop-area'),
      oStatNbFiles = document.getElementById('nb_files'),
      oListFiles = document.getElementById('list_files');
  var iNbFiles = 0;

```

Notez que nous avons préparé un élément de type liste ([UL](#)) dont les lignes seront construites dynamiquement au moment de réceptionner les fichiers.

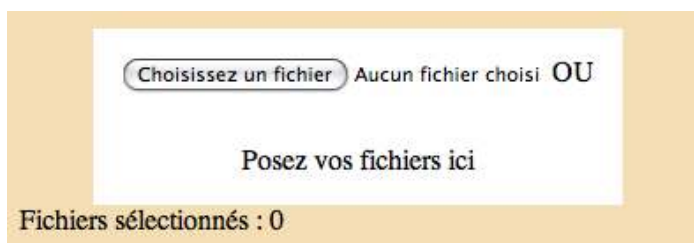


Figure 8.5 — Avec un peu de CSS et un goût inné pour le design, on obtient ça

8.3.2 Savoir écouter

Nous allons maintenant écouter les interactions de l'utilisateur avec notre zone de drop et le sélecteur de fichiers. Le but dans les deux cas étant d'obtenir la [FileList](#) que nous allons passer à une méthode que nous définirons un peu plus tard : [XHR2Uploader.addNewFiles\(\)](#).

Récupération de la liste de fichiers

```

// Lors de la sélection, on donne la liste des fichiers à la fonction
oInput.onchange = function() {
  XHR2Uploader.addNewFiles( this.files );
};
// dragover doit toujours être annulé
oDropArea.ondragover = function(e) {
  e.dataTransfer.effectAllowed = 'copy';
  e.dataTransfer.dropEffect = 'copy';
  e.preventDefault();
};
// Lors du jeté de fichier, on donne la liste à la fonction
oDropArea.ondrop = function(e) {
  // on détecte si le navigateur sait accéder aux fichiers
  if('files' in e.dataTransfer) {

```

```

        XHR2Uploader.addNewFiles( e.dataTransfer.files );
    } else {
        oDropArea.innerHTML = 'Navigateur incapable';
        oStatNbFiles.innerHTML = 'Un certain nombre';
    }
};

```

8.3.3 Savoir donner

Nous allons ranger dans un même espace de nom (ici `XHR2Uploader`) les méthodes relatives à l'utilisation de XHR2, vous pourrez toujours réutiliser ce code comme base pour en faire une classe adaptable à vos projets.

Dans cette première méthode, appelée par nos événements d'interface, on récupère une collection d'objets `File`, on crée la liste des fichiers sélectionnés, ce qui inclue l'élément `progress` et si aucune XHR n'est en cours, on exécute la méthode `XHR2Uploader.startUpload`.

Méthode de réception des fichiers

```

var XHR2Uploader = {
    aQueue:[], // contient la liste des objets File à envoyer
    oXHR:new XMLHttpRequest(),
    oCurrentFile:null, // pointeur utile vers le fichier à envoyer
    addNewFiles: function(aFiles) {
        // ici on accède à la liste des fichiers sélectionnés (FileList)
        // mise à jour du nombre de fichiers
        iNbFiles += aFiles.length;
        oStatNbFiles.innerHTML = iNbFiles;
        // variables locales utilisées dans notre boucle
        var oLine, oProgress, oFile;
        // On boucle sur la collection FileList pour
        // récupérer le nom et la taille de chaque fichier
        for(var i = 0; i < iNbFiles; i++) {
            oFile = aFiles[i]; // référence locale
            // ajout à la liste des fichiers à traiter
            XHR2Uploader.aQueue.push(oFile);
            // liste des fichiers mis à jour dans l'interface
            oLine = document.createElement('LI');
            oLine.innerHTML = oFile.name + ' ('+oFile.size+' octets) ';
            // affichage de la barre de progression
            oProgress = document.createElement('PROGRESS');
            // ceci nous servira à identifier la ligne du fichier
            oProgress.id = oFile.name + oFile.size;
            oProgress.max = oFile.size;
            oProgress.value = 0;
            oLine.appendChild(oProgress);
            oListFiles.appendChild(oLine);
        }
        // si la requête n'a jamais été lancée ou si la requête précédente est
        terminée
        if(XHR2Uploader.oXHR.readyState === 0
            || XHR2Uploader.oXHR.readyState === 4) {

```

```

        XHR2Uploader.startUpload();
    }
},

```

Voici ensuite les méthodes qui :

- font l'envoi effectif des fichiers au serveur, en utilisant `FormData`,
- mettent à jour la barre de progression pendant et à la fin de l'*upload*,
- gèrent la queue d'envoi.

Suite et fin de la gestion de l'envoi via XHR2

```

// à appeler lorsqu'aucun envoi n'est en cours
startUpload:function() {
    // reste-t-il quelque chose à envoyer ?
    if(XHR2Uploader.aQueue.length < 1)
        return;
    // récupération des infos du formulaire
    XHR2Uploader.oXHR.open(oInput.form.method, oInput.form.target);
    // le constructeur de FormData se trouve dans le scope global
    var oFormData = new FormData( );
    // pointeur vers le premier fichier de la queue
    XHR2Uploader.oCurrentFile = XHR2Uploader.aQueue.shift();
    // on construit l'équivalent du formulaire HTML
    oFormData.append(
        oInput.name, // récupération du nom du champ
        XHR2Uploader.oCurrentFile
    );
    // XHR2.send va formater une requête POST classique
    // à partir de cet objet
    XHR2Uploader.oXHR.send( oFormData );
},
// appelée pendant l'envoi : met à jour les barres de progression
onUploading:function(e) {
    // on récupère la barre de progression associée à notre fichier
    var oProgress =
        document.getElementById( XHR2Uploader.oCurrentFile.name +
        XHR2Uploader.oCurrentFile.size);
    // mise à jour de la valeur, le navigateur s'occupe du reste
    oProgress.value = e.loaded;
    // mise à jour du texte, pour les navigateurs sans le support
    oProgress.innerHTML =
        Math.round(
            (oProgress.value / oProgress.max) * 100)+'%';
},
// appelée à la fin de l'envoi : mise à jour de l'interface,
// poursuit la queue d'envoi
onUploaded:function(e) {
    var oProgress = document.getElementById(
        XHR2Uploader.oCurrentFile.name + XHR2Uploader.oCurrentFile.size);
    // navigateur avec support
    oProgress.value = XHR2Uploader.oCurrentFile.size;
    // navigateurs sans support
    oProgress.innerHTML = 'OK !';
    XHR2Uploader.startUpload(); // fichier suivant
}

```

```
}; // fin du namespace XHR2Uploader  
// on écoute l'évolution de la requête d'envoi  
XHR2Uploader.oXHR.upload.onprogress = XHR2Uploader.onUploading;  
XHR2Uploader.oXHR.onload = XHR2Uploader.onUploaded;  
</script>
```



Tester le résultat !

<http://www.livre-html5.com/drop/select-multiple-file-api-xhr2-progress.php>

Résultat

Comme prévu, sur Firefox 4, Chrome et Safari, vous pouvez sélectionner plusieurs fichiers à partir du formulaire ou bien prendre des fichiers sur le bureau et les glisser dans la zone de la page prévue à cet effet. Vous verrez alors une barre de progression, native ou textuelle, vous indiquer le pourcentage d'octets envoyés.

Ce code a en plus l'avantage de maximiser la taille des fichiers qui peuvent être envoyés puisque ceux-ci sont envoyés individuellement et non plus dans la même requête POST.

Enfin comme il n'y a pas le rechargement de page classique, même durant l'envoi vous pouvez rajouter d'autres fichiers, ce qui permet de mettre à parti les longues heures d'upload pour trouver encore plus de médias à envoyer !

8.4 ALLER PLUS LOIN

8.4.1 Gérer le curseur pendant le drop de fichiers

Concentrons-nous un peu sur l'ergonomie utilisateur. Au moment où celui-ci survole notre page avec son fichier, il a un retour visuel de sa souris qui lui tient à peu près ce langage « tu peux jeter ton fichier où tu veux sur cette page ». Et si il l'écoute, le fichier s'affiche en lieu et place de la dite page, ce qui n'est pas souhaitable. L'idée ici est :

- lorsque l'utilisateur survole la page avec son fichier, la zone de *drop* doit attirer son attention ;
- il ne doit pas avoir l'impression qu'il peut jeter son fichier n'importe où sur la page ;
- si il le fait, il ne faut pas le pénaliser en laissant le navigateur afficher l'image (ce qui lui fait faire appuyer sur bouton précédent, et donc recharger la page) ;
- au survol de la zone de *drop*, il doit avoir la confirmation visuelle que c'est le bon endroit.

Nous allons donc compléter notre code en :

1. rajoutant la gestion des mêmes évènements sur l'élément hiérarchique le plus haut du DOM : `document.body`,
2. rajoutant une classe sur `document.body` qui nous permette de styler de manière ostentatoire la zone de *drop*,
3. rajoutant une classe sur la zone de *drop* au moment où l'utilisateur passe au dessus,
4. déplaçant l'endroit où on gère la réception du fichier,
5. enlevant les classes lorsque l'utilisateur quitte la zone de *drop*, la page, ou a fini son opération.

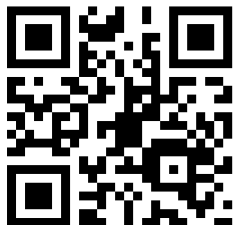
Code complet de la gestion du curseur natif

```
<div id="droparea">
  DROP demo
</div>
<style>
/* isolons notre DIV cobaye au milieu de la page */
#drop-zone {
  background-color:gray;
  width:100px;
  margin:0 auto;
  height:50px;
  text-align:center;
  line-height:50px;
}
/* coloration de la DIV cobaye au moment d'entrer sur la page */
.dragging #drop-zone{
  border:5px dotted red;
}
/* changement de couleur lorsque l'utilisateur est au-dessus de la DIV */
#drop-zone.dragover {
  border-color:green;
}
</style>
</script>
// on commence par annuler toutes les réactions standard sur toute la page
document.body.ondragover = function(e) {
  e.preventDefault(); // pas de curseur affiché sur la page
};
document.body.ondrop = function(e) {
  e.preventDefault(); // si l'utilisateur jette son fichier n'importe où, on
empêche le navigateur de l'afficher
  document.body.className = ''; // notre DIV redevient banale
};
// déclenché lorsque l'utilisateur arrive la première fois au-dessus de la page
document.body.ondragenter = function() {
  document.body.className = 'dragging';// on rend notre DIV plus sexy (question
de goûts)
};
// déclenché lorsque l'utilisateur quitte la page
document.body.ondragleave = function() {
  document.body.className = '';// notre DIV redevient banale
```

```

});
// référence à notre zone de lâcher de fichier
var oDropArea = document.getElementById('drop-zone');
// plus besoin d'annuler cet événement, il est annulé plus haut
oDropArea.ondragover = function(e) {
    e.dataTransfer.effectAllowed = 'copy'; // correction du bug Mozilla #610378
    e.dataTransfer.dropEffect = 'copy'; // changement du style du curseur,
    l'utilisateur est en confiance
};
// réception du fichier
oDropArea.ondrop = function(e) {
    // notez l'absence d'annulation de l'évènement, annulé plus haut dans le DOM
    if('files' in e.dataTransfer) {
        oDropArea.innerHTML = 'Vous aviez '+
            e.dataTransfer.files.length
            +' fichier(s)';
    } else {
        oDropArea.innerHTML = 'Pas d'accès fichier';
    }
    oDropArea.className = ''; // notre DIV redevient banale
};
oDropArea.ondragenter = function() {
    // notre DIV déjà sexy réagit aux effleurements de souris
    oDropArea.className = 'dragover';
};
oDropArea.ondragleave = function() {
    // notre DIV redevient juste sexy
    oDropArea.className = '';
};
</script>

```



Voir la démo !

<http://www.livre-html5.com/drop/drop.php>

8.4.2 Styler le sélecteur de fichiers

Le sélecteur de fichiers natif n'est pas spécialement joli, cependant il est reconnaissable aisément, aussi pour des raisons d'ergonomie il vaut mieux ne pas le déguiser. Mais si votre politique est le *pixel-perfect*, vous préférerez peut-être avoir un sélecteur de fichiers qui soit **le même pour la version flash et la version native, et même sur tous les navigateurs et systèmes.**

Pour la version Flash du sélecteur multiple, choisissez une librairie qui permet d'avoir l'objet flash transparent (YUI 3 : *Uploader* par exemple), afin de le déplacer au-dessus de votre bouton. La version HTML marche exactement sur le même principe :

1. créez et stylez votre bouton,
2. insérez votre `input type="file"`,

3. en CSS positionnez-le au-dessus du bouton, dimensionnez-le,
4. donnez-lui une opacité de 0 !

Ca s'appelle du *click hijacking*, et c'est généralement utilisé par des vilains pirates ou des publicitaires aux abois pour tromper l'utilisateur. Mais vous êtes un professionnel honnête avec un problème de graphisme à résoudre, donc vous pouvez appeler ça « une astuce ».

Suggestion de code

```
<form>
  <button type="button">Beau bouton de sélection de fichiers</button>
  <input type="file" multiple accept="image/*,video/*" />
</form>
<style>
form {
  display:block;
  height:10em;
  width:20em;
  line-height:10em;
  background-color:gray;
  text-align:center;
}
input[type="file"] {
  opacity: 0;
  /* alignage sur le bouton visible*/
  top: 7em;
  position: absolute;
  left:0;
  /* dimensions imposantes pour être sûr de ne pas louper le clic */
  width:100%;
  height:2.5em;
  cursor:pointer;
}
/* stylage libre du bouton */
button[type="button"] {
  border:0;
  display: inline-block;
  padding: 5px 10px 6px;
  .....
}
</style>
```



Voir la démo !

<http://www.livre-html5.com/drop/styler-select.php>

La technique marche sur tous les navigateurs, mais bien sur le code CSS3 et les sélecteurs CSS2 utilisés ici pour styler le bouton visible sont à adapter en fonction des navigateurs que vous ciblez.

8.5 ON FAISAIT COMMENT AVANT ?

Si l'évolution vers HTML5 peut se faire en douceur en n'introduisant ces fonctionnalités que pour les navigateurs hors Internet Explorer, nous ne saurions être complets sans rappeler les méthodes pour obtenir le même résultat

Notez bien que ces méthodes sont tellement différentes de HTML5 que le travail d'intégration au front s'en trouve doublé. L'utilisation de la technologie native permet d'obtenir une expérience utilisateur beaucoup plus fluide et donc engageante : concrètement on constate que les utilisateurs à qui l'on sert ces *uploaders* natifs envoient plus d'images que les autres.

À vous donc de voir en connaissance de cause si l'investissement en temps en vaut la peine.

8.5.1 La sélection multiple et la progression avec Flash

La sélection multiple et les indications de progression d'envoi peuvent s'obtenir avec Flash ou Java. Flash étant plus répandu et malgré tout moins lourd à l'exécution que Java, privilégiez-le.

Il y a des dizaines de *widgets*, préférez celles des grandes bibliothèques JavaScript (YUI, jQuery UI) qui sont plus solides et plus modulaires. Elles vous permettent normalement d'accéder en JavaScript aux événements intéressants, et vous fournissent des instructions pour installer le fichier `.swf` sur votre serveur, qui se chargera du transport du fichier, pendant que vous pourrez coder toute l'interface et les événements de manière traditionnelle en HTML / CSS / JavaScript.

Normalement, les événements et informations fournies sont très proches dans leur structure de la version HTML5 que nous avons vue :

- la fenêtre de sélection native permet de sélectionner plusieurs fichiers,
- limitation par type de fichier,
- accès à tous les fichiers sélectionnés, notamment le nom et la taille,
- pendant *l'upload*, mise à jour régulière du nombre de bytes envoyés,
- événements de fin avec réponse du serveur.

Vous pouvez donc envisager sereinement de coder une façade JavaScript pour abstraire la méthode de transport du fichier, et fournir la même interface utilisateur quel que soit le navigateur.

8.5.2 Le drop avec des applets Java

Au contraire du *plugin* Flash, Java est extérieur au navigateur. Concrètement il positionne son interface physiquement au-dessus du navigateur dans l'OS, réceptionnant donc les fichiers que vous lancez dans la zone. Ensuite *via* JavaScript il notifie à la page les événements intéressants comme le début et la fin de l'envoi d'un fichier et la réponse serveur.

Les désavantages sont nombreux :

- il est probable que vous n'ayez pas les compétences en interne en Java, et un développement coûterait de toute façon cher. Il vaut donc mieux utiliser une librairie ;
- ces librairies sont généralement payantes. C'est une petite jungle d'offres aux prix et surtout à la qualité variés. Tapez « *drag and drop upload java applet* » dans votre moteur de recherche et pleurez ;
- il peut y avoir un message d'avertissement intimidant pour l'utilisateur, et à tout le moins il faut au moins installer une fois l'applet ;
- sur d'anciennes machines ou sur les Mac OS, exécuter la machine virtuelle Java est vraiment pénible et ralentit l'expérience utilisateur ;
- si vous êtes un tout petit peu paranoïaque : vous ne saurez jamais vraiment si vous n'êtes pas en train de faire installer un virus à vos utilisateurs ;
- l'applet n'utilise pas les cookies du navigateur : si votre *upload* se fait sur une partie privée, il faut que l'*applet* ait prévu un mécanisme vous permettant d'y injecter votre cookie ou que vous bricoliez l'URL d'envoi en y rajoutant un *hash* du cookie...
- la compatibilité d'une *applet* Java est douteuse : certaines personnes ont des installations de la machine Java qui ne sont pas forcément compatibles avec votre librairie ;
- *antivirus* et *firewall* peuvent très bien empêcher l'applet de fonctionner ;
- en moyenne, 20 % des utilisateurs n'ont pas Java installé.

Malgré cela, on constate que les utilisateurs à qui l'on fournit cette solution de drop de fichiers l'utilisent de manière privilégiée, et sont un peu plus engagés que les autres ! Il semblerait donc que compatibilité, message d'alerte et lourdeur d'exécution n'embêtent finalement que les développeurs...

Cette seule constatation peut vous encourager à utiliser cette technologie malgré tout, car hormis HTML5 c'est la seule qui permet l'accès à cette fonctionnalité.

En résumé

Nous avons vu comment construire une fonctionnalité qui, il y a trois ans à peine, n'était possible qu'avec des *plugins*. Malgré cela nous avons vu qu'avec un travail supplémentaire, la rétrocompatibilité pouvait être assurée.

Cet *uploader* nous a également démontré la puissance des standards du Web : l'interopérabilité entre navigateurs récents est assurée malgré la jeunesse de toutes ces spécifications et le fait qu'elles ne soient pas encore toutes au stade de la recommandation. Elles sont toutes prévues pour s'articuler entre elles, et nous aurions pu rajouter Canvas pour faire un éditeur de photo, WebSocket pour rajouter de la collaboration, la Audio API pour faire un éditeur de musique... bref, les développeurs web mettent réellement le pied dans le monde de l'appliatif.

9

La géolocalisation et l'API Google Maps

Objectif

Il existe une différence majeure entre un navigateur d'ordinateur fixe et un navigateur mobile : le navigateur mobile peut être utilisé n'importe où ! De cette différence vient bien sûr une nouvelle utilisation applicative. La plus évidente étant l'utilisation du GPS sur une carte pour arriver à se localiser.

Cette nouvelle fonctionnalité est largement utilisée par les applications natives sur mobile telles que Google Maps, Foursquare, ou le partage d'un lieu sur Facebook et Twitter. HTML5 apporte cette utilisation aux navigateurs des smartphones grâce à une API très simple à utiliser, dont nous allons voir le fonctionnement en détail au cours de ce chapitre.

9.1 INTRODUCTION

9.1.1 Exemple d'application : Petit Poucet part en randonnée

Vous venez tout juste d'arriver sur votre lieu de vacances, et vous allez enfin pouvoir vous livrer à votre activité favorite : la randonnée ! Vous vous rendez donc au point de départ de votre parcours, et vous apercevez qu'il est déjà tard et que la nuit va probablement tomber d'ici une ou deux heures.

« Si je ne reconnais pas le trajet que j'ai suivi à l'aller, je risque de me perdre sur le chemin du retour, je devrais prendre des précautions ».

Il se trouve que par chance, vous connaissez un site – pardon, une « web app » – qui propose justement de sauvegarder votre parcours sur une carte afin de pouvoir trouver facilement le chemin du retour ! Vous faites donc un petit tour par votre navigateur, sans rien installer, et c'est parti !

Au fur et à mesure de votre progression, votre position est constamment mise à jour sur la carte de l'application qui vous indique également où est le nord et quelle est votre vitesse actuelle. Pas mal pour une simple page web non ?

Nous allons réaliser au cours de ce chapitre une application de ce type, tout d'abord en voyant comment récupérer les informations de géolocalisation, puis nous utiliserons l'API Google Maps pour utiliser ces informations sur une carte.

9.1.2 La compatibilité

Avant de commencer, l'éternelle question de la compatibilité actuelle des navigateurs se pose. Firefox, Safari, Opera, Chrome et Internet Explorer 9 proposent cette fonctionnalité. En revanche, Internet Explorer 6, 7 et 8 ne la supportent pas (pour lesquels il faudra toujours compter sur la géolocalisation par adresse IP). Mais est-ce bloquant pour autant ?

Les applications utilisant la géolocalisation pour mobiles et pour ordinateurs de bureau sont très différentes. Et à moins que vous ne comptiez vous balader dans la rue avec votre unité centrale reliée à une antenne GPS et un groupe électrogène dans une brouette, vous ne vous déplacerez pas avec votre navigateur fixe.

Bon alors certes il y a quelques cas d'utilisation particuliers avec les ordinateurs portables connectés à Internet pendant les trajets en train ou en bateau, mais honnêtement, la principale cible de cette API de géolocalisation ce sont les smartphones. Et devinez quoi ? Les navigateurs iPhone, Android et Blackberry supportent la géolocalisation !

9.1.3 Les services de géolocalisation

Au cours de ce chapitre nous allons parler de mystérieux « services de géolocalisation ». De quoi s'agit-il ? Un navigateur qui souhaite implémenter la géolocalisation va utiliser un service externe pour récupérer votre position. Firefox, Google Chrome ou encore Opera utilisent les services Google Location. Internet Explorer 9 utilise le service de géolocalisation de Microsoft, et il semblerait qu'Apple ait réussi à monter son propre réseau notamment grâce à tous ses mobiles qui se baladent dans la nature et qui renvoient en permanence des infos sur l'environnement.

Ces systèmes font de la triangulation à partir de la force des signaux Wifi que capte votre ordinateur ou votre mobile, et utilisent dans le cas de Google une cartographie de ces réseaux enregistrée par les Google Cars. En clair : les zones où ce service est précis correspondent aux zones visibles par le « Street View » de Google Maps. Ailleurs, c'est l'IP qui est utilisée.

En bref : en utilisant la géolocalisation HTML5 votre navigateur fait appel à une entité externe qui va vous renvoyer votre position, et ce service est appelé « service de géolocalisation ».

9.2 RÉCUPÉRER LES COORDONNÉES

9.2.1 Tester le support

Tout d'abord, nous allons tester le support de la géolocalisation par le navigateur de l'utilisateur. Il suffit pour cela de tester l'existence de `navigator.geolocation` :

Tester le support

```
if (navigator.geolocation)
    alert("Géolocalisation supportée");
else
    alert("Géolocalisation non supportée");
```

C'est d'ailleurs cet objet `navigator.geolocation` qui va nous servir à faire appel aux méthodes et aux attributs de géolocalisation.

9.2.2 Récupérer la position

La méthode la plus importante que propose l'objet `navigator.geolocation` est bien entendu celle qui consiste à récupérer les coordonnées de la position de l'utilisateur. Il s'agit de `navigator.geolocation.getCurrentPosition()`, qui prend en premier paramètre la fonction à déclencher lorsque notre page aura obtenu la réponse du service de géolocalisation.

Le temps de réponse du service étant imprévisible, il est obligatoire de passer par ce type de fonction appelée *callback*, qui se déclenchera automatiquement lorsque nous aurons obtenu la réponse du service. La fonction de *callback* prend en paramètre le nom de la variable dans laquelle seront injectées les coordonnées fournies par le service de géolocalisation. Une fois ces coordonnées obtenues on y a accès par la propriété `coords.latitude` et `coords.longitude`.

Ainsi on peut simplement tester la réception des coordonnées par les quelques lignes suivantes.

Réception des coordonnées

```
// on teste le support
if (navigator.geolocation) {
    // si la géolocalisation est supportée, on récupère la position
    navigator.geolocation.getCurrentPosition(function(position) {
        // on affiche la latitude et la longitude
        alert("Latitude : " + position.coords.latitude +
            ", Longitude : " + position.coords.longitude);
    });
}
```

```
} else {  
    alert("Géolocalisation non supportée");  
}
```

Dans cet exemple simple nous utilisons une fonction anonyme pour traiter le cas de succès, mais il est préférable d'utiliser des fonctions séparées afin de garder notre code compréhensible si celui-ci se complexifie, comme nous le verrons dans la prochaine partie.

9.2.3 Les demandes d'autorisation

Lorsque la méthode `getCurrentPosition()` est exécutée, un bandeau apparaît en haut de page afin de demander à l'utilisateur s'il souhaite partager sa position.

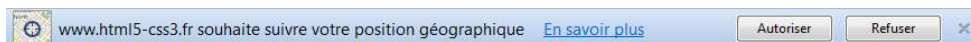


Figure 9.1 — Demande d'autorisation de l'accès à la position sur Chrome.



Figure 9.2 — Demande d'autorisation de l'accès à la position sur iPhone.

Si l'utilisateur accepte, alors `getCurrentPosition()` va faire appel au service de géolocalisation pour obtenir sa position. S'il refuse, une fonction de *callback* d'erreur est déclenchée.

9.3 GÉRER LES ERREURS

9.3.1 Les différents cas d'échec

La méthode `getCurrentPosition()` peut prendre un second paramètre, qui est la fonction de callback en cas d'erreur :

```
getCurrentPosition(successCallback, errorCallback) ;
```

Le *callback* d'erreur est déclenché si :

- l'utilisateur refuse de partager sa position,
- le service ne parvient pas à déterminer la position de l'utilisateur,
- le service ne répond pas assez vite.

Afin de savoir pour quelle raison le callback d'erreur a été déclenché, nous pouvons interroger attribut `code` de l'objet obtenu en paramètre du callback, qui peut prendre pour valeur trois constantes :

- `PERMISSION_DENIED`
- `POSITION_UNAVAILABLE`
- `TIMEOUT`

Ces trois constantes correspondent respectivement aux trois cas cités précédemment.

9.3.2 Exemple complet de gestion des erreurs

L'extrait de code suivant illustre l'utilisation du *callback* de réussite, du *callback* d'erreur, ainsi que la gestion des codes d'erreur.

Gestion des erreurs

```
// on nomme les 2 fonctions de callback utilisées par getCurrentPosition
navigator.geolocation.getCurrentPosition(successCallback, errorCallback);
// la fonction de réussite affiche les coordonnées contenues dans "position"
function successCallback(position){
    alert("Latitude : " + position.coords.latitude +
        ", Longitude : " + position.coords.longitude);
};
// la fonction d'échec teste l'erreur contenue dans "error"
function errorCallback(error){
    // un message d'erreur est affiché selon le type d'erreur
    switch(error.code){
        case error.PERMISSION_DENIED:
            alert("L'utilisateur n'a pas autorisé l'accès à sa position");
            break;
        case error.POSITION_UNAVAILABLE:
            alert("L'emplacement de l'utilisateur n'a pas pu être déterminé");
            break;
        case error.TIMEOUT:
            alert("Le service n'a pas répondu à temps");
            break;
    }
};
```

D'un point de vue de l'ergonomie de l'application, il est important de prévoir un comportement pour tous les cas d'erreurs possibles. Si l'utilisateur refuse l'accès à sa position, ce n'est pas la peine de le harceler de messages génériques d'erreur l'invitant à réessayer de se connecter plus tard...

9.4 TRANSFORMER VOTRE APPLICATION EN VÉRITABLE GPS

9.4.1 Les options

La fonction `getCurrentPosition()` peut également prendre des options en troisième paramètre. Ces options sont :

- `enableHighAccuracy` : qui permet d'activer l'utilisation du GPS si l'appareil en est équipé pour une meilleure précision (au prix d'un délai un peu plus long),
- `timeout` : qui sert à déclarer la durée maximale que peut prendre l'appel au service,
- `maximumAge` : qui sert à spécifier la durée de validité d'une position pendant laquelle elle sera gardée en cache.

Les valeurs `timeout` et `maximumAge` sont à indiquer en millisecondes.

Voici un exemple d'utilisation du paramètre d'options.

Exemple d'utilisation du paramètre d'options

```
navigator.geolocation.getCurrentPosition(
    successCallback,
    errorCallback,
    { enableHighAccuracy : true,
      timeout : 10000,
      maximumAge : 600000
    });
```

Dans cet exemple on établit la durée maximale de l'appel au service à 10 secondes et une durée de validité d'une position à 10 minutes.

La valeur par défaut de `timeout` est `Infinity` et celle de `maximumAge` est 0, c'est-à-dire que l'application ne définit pas de durée maximale pour l'appel au service de géolocalisation et qu'elle demandera systématiquement une mise à jour de la position.

9.4.2 Suivre les déplacements

Il peut arriver que l'on veuille suivre les déplacements d'un utilisateur, comme c'est le cas pour l'application que nous souhaitons réaliser. Pour cela nous n'allons pas lancer régulièrement des appels au service de géolocalisation à la main, mais utiliser une méthode existante très pratique : `watchPosition()`.

`watchPosition()` s'utilise exactement de la même manière que `getCurrentPosition()`. Elle prend les trois mêmes paramètres mais contrairement à `getCurrentPosition()`, elle retourne un identifiant unique de suivi de position.

Utilisation de `watchPosition`

```
var iWatchId = navigator.geolocation.watchPosition(successCallback,
    errorCallback);
```

Le suivi de la position peut être à tout moment interrompu grâce à la fonction `clearWatch()`, qui prend en paramètre l'identifiant du suivi.

Arrêt du suivi de la position

```
navigator.geolocation.clearWatch(iWatchId);
```

En utilisant `watchPosition()` au lieu de `getCurrentPosition()`, le service de géolocalisation va en permanence fournir les coordonnées de l'utilisateur en déclenchant les *callbacks*. Nous sommes ensuite libres d'utiliser ces coordonnées comme bon nous semble dans un traitement JavaScript correspondant à nos besoins.

9.4.3 Mise en pratique

Nous allons réaliser une petite application qui affiche les coordonnées de la position de l'utilisateur au fur et à mesure de ses déplacements.

Structure HTML de notre application

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
  Latitude : <span id="latitude"></span><br />
  Longitude : <span id="longitude"></span><br />
  Erreur : <span id="error"></span>
</body>
</html>
```

Nous allons simplement effectuer le suivi de la position grâce à `watchPosition()` et afficher les résultats obtenus dans les labels correspondants.

Script de mise à jour des labels avec les coordonnées

```
// on récupère nos trois labels
var oLatitude = document.getElementById("latitude"),
    oLongitude = document.getElementById("longitude"),
    oErrors = document.getElementById("error");
// on lie watchPosition à nos callbacks et on active le GPS
navigator.geolocation.watchPosition(successCallback,
                                     errorCallback,
                                     {enableHighAccuracy:true});

// en cas de réussite on affiche les coordonnées
function successCallback(position){
  oLatitude.innerHTML = position.coords.latitude;
  oLongitude.innerHTML = position.coords.longitude;
  oErrors.innerHTML = "";
};

// en cas d'échec on affiche un label d'erreur
function errorCallback(error){
  oLatitude.innerHTML = "";
  oLongitude.innerHTML = "";
  oErrors.innerHTML = "Erreur";
};
```

Comme il est assez inutile d'effectuer le suivi du déplacement d'un ordinateur fixe, `watchPosition()` est surtout destinée aux déplacements des *smartphones*, en particulier ceux équipés d'un GPS. C'est pourquoi nous avons activé ce dernier avec le paramètre d'option : `enableHighAccuracy:true`; afin que vous puissiez tester cette application dans la rue. Grâce à cet exemple nous avons pu voir à quel point il est simple de récupérer la latitude et la longitude d'un utilisateur. La latitude et la longitude c'est déjà une bonne chose, mais que peut-on avoir de plus grâce à notre objet `position` ?

9.4.4 La vitesse, la boussole et l'altitude

L'objet `Position` que l'on obtient grâce à `getCurrentPosition()` ou `watchPosition()` contient bien plus que la latitude et la longitude. On peut ainsi également récupérer :

- `position.timestamp` : l'heure à laquelle a été mise à jour la position,
- `position.coords.altitude` : l'altitude en mètres de l'utilisateur,
- `position.coords.accuracy` : le rayon d'imprécision en mètres des coordonnées,
- `position.coords.altitudeAccuracy` : la précision de l'altitude,
- `position.coords.heading` : l'angle compris entre 0 et 360° par rapport au Nord,
- `position.coords.speed` : la vitesse en mètres par seconde de l'utilisateur par rapport à sa dernière position.

Notez que certaines de ces informations ne sont disponibles que si le GPS est activé, et qu'il est possible que plusieurs mises à jour de la position soient nécessaires afin d'obtenir des résultats pertinents (dans le cas de la boussole et de la vitesse par exemple). De plus, l'angle par rapport au Nord que fournit le service de géolocalisation n'est pas aussi précis que la boussole interne d'un *smartphone*. Mais il est déjà très utile dans le cadre de comportements stables de déplacement tels que le suivi du trajet d'une voiture par une application de type GPS.



Tester l'application !

<http://www.livre-html5.com/geolocalisation/text.php>

9.5 UTILISER L'API GOOGLE MAPS

Cette partie du chapitre ne fait pas du tout partie de la spécification HTML5 et n'a pour utilité que de mettre en pratique la géolocalisation en apportant une représentation un peu plus sympathique que de simples lignes de texte. Nous allons donc tout d'abord intégrer l'API JavaScript Google Maps afin d'afficher une carte, puis placer un marqueur à l'emplacement de l'utilisateur sur cette carte. Nous verrons

enfin comment tracer une ligne qui suit le parcours d'un utilisateur qui se déplace dans le cadre de notre application de randonnée.

9.5.1 Intégrer l'API Google Maps

Pour inclure les fonctions Google Maps dans notre application, il faut ajouter une balise `script` dans la partie `head` de notre page, qui pointe sur l'URL de l'API Google Maps. L'API Google Maps est disponible sous plusieurs déclinaisons, que l'on peut choisir en ajoutant un paramètre à l'URL. Le seul paramètre qui nous intéresse dans le cadre de notre application est le paramètre `sensor`, qui permet de spécifier que nous utilisons des coordonnées GPS.

Intégration de l'API Google Maps

```
<script src="http://maps.google.com/maps/api/js?sensor=true"></script>
```

Nous sommes maintenant prêts à utiliser les méthodes fournies par l'API Google Maps.

9.5.2 Création de la page qui va contenir la carte

Tout d'abord, nous avons besoin d'un conteneur pour notre carte, auquel nous attribuons l'id `map_canvas` :

```
<div id="map_canvas"></div>
```

Nous voulons que notre carte occupe tout l'espace disponible dans la page. Google préconise le CSS suivant afin de maximiser la compatibilité de l'affichage plein écran :

```
html { height: 100% }  
body { height: 100%; margin: 0px; padding: 0px }  
#map_canvas { width: 100%; height: 100% }
```

Enfin il est possible de spécifier aux smartphones que notre page ne doit pas être redimensionnable grâce à la balise `meta viewport` :

```
<meta name="viewport" content="initial-scale=1.0, user-scalable=no" />
```

Voici donc à quoi ressemble notre page HTML pour le moment :

Structure HTML d'accueil de la carte Google Maps

```
<!DOCTYPE html>  
<html>  
<head>  
  <meta name="viewport" content="initial-scale=1.0, user-scalable=no" />  
  <style type="text/css">  
    html { height: 100% }  
    body { height: 100%; margin: 0px; padding: 0px }
```

```
#map_canvas { height: 100% ; width:100%;}  
</style>  
<script src="http://maps.google.com/maps/api/js?sensor=true"></script>  
</head>  
<body>  
  <div id="map_canvas"></div>  
</body>  
</html>
```

Une fois cette structure mise en place il va maintenant falloir charger la carte dans notre conteneur.

9.6 AFFICHER LA CARTE GOOGLE MAPS

9.6.1 Initialiser la carte

Il est temps de passer aux choses sérieuses ! Une fois notre page HTML chargée nous devons initialiser la carte via JavaScript. L'objet de l'API Google Maps qui nous intéresse est l'objet `google.maps.Map`, dont le constructeur prend en paramètres :

- l'élément du DOM dans lequel injecter la carte,
- un ensemble de paramètres d'initialisation. Nous utiliserons :
 - `zoom` pour spécifier le niveau de zoom,
 - `center` qui correspond aux coordonnées sur lesquelles sera centrée la vue,
 - `mapTypeId` qui sert à indiquer le type d'affichage souhaité (plan, satellite...).

Nous fournissons donc au constructeur de `Map` :

- notre conteneur ayant pour `id` `map_canvas`,
- un niveau de zoom assez précis de 19, afin de voir facilement la carte se déplacer,
- un centre de coordonnées 48.858565, 2.347198, qui correspondent au centre de Paris,
- le type de carte `MapTypeId.ROADMAP`, qui est l'affichage de type plan.

Initialiser la carte Google Maps

```
<script>  
  // on ajoute un listener de fin de chargement de la page  
  window.addEventListener('load', function() {  
    // on construit l'objet Map dans notre conteneur "map_canvas"  
    oMap = new google.maps.Map(document.getElementById("map_canvas"), {  
      // on utilise les options prévues  
      zoom: 19,  
      center: new google.maps.LatLng(48.858565, 2.347198),  
      mapTypeId: google.maps.MapTypeId.ROADMAP  
    });  
  }, false);  
</script>
```

On remarque que les coordonnées du `center` sont fournies au travers d'un nouvel objet : `google.maps.LatLng` que l'on devra utiliser à chaque fois que l'on voudra utiliser des coordonnées. Il est maintenant temps d'ouvrir notre page HTML dans un navigateur ! Si tout se passe bien, vous devriez avoir obtenu quelque chose comme la figure 9.3.



Figure 9.3 — Notre première Google Map !

Si vous testez sur un *smartphone*, assurez-vous de ne pas avoir oublié la balise `meta viewport` afin que les boutons s'affichent de la bonne taille.

Bon, nous avons une carte qui s'affiche, c'est déjà pas mal, mais ça manque cruellement de géolocalisation ! Nous allons maintenant mettre en pratique ce que nous avons vu au début de ce chapitre afin de géolocaliser la position de l'utilisateur et afficher sa position sur la carte.

9.6.2 Ajouter la géolocalisation HTML5

Puisque nous souhaitons réaliser une application qui va effectuer le suivi des déplacements de l'utilisateur, nous allons utiliser la méthode `watchPosition()` plutôt que `getCurrentPosition()`.

En théorie nous devrions gérer les cas où la géolocalisation n'est pas supportée ainsi que les trois cas d'erreur possibles vus dans le chapitre précédent mais afin de ne pas alourdir le code nous allons aller droit au but pour faciliter la compréhension. Nous n'utiliserons donc pas de callback d'erreur pour le moment et nous nous contenterons donc de cet appel à `watchPosition()` :

```
navigator.geolocation.watchPosition(successCallback, null,  
                                     {enableHighAccuracy:true});
```

Afin de pouvoir tester les déplacements de notre randonneur, nous activons les données GPS avec `enableHighAccuracy`. Nous voici donc à nouveau au cœur de la méthode `successCallback()`, dans laquelle nous allons récupérer les coordonnées de position dans l'objet `google.maps.LatLng` :

```
oCoords = new google.maps.LatLng(position.coords.latitude,  
                                   position.coords.longitude);
```

9.6.3 Centrer la carte sur un marqueur

Il suffit ensuite de passer ces coordonnées à la méthode `panTo()` de notre objet `Map` pour centrer la carte en ce point :

```
oMap.panTo(oCoords);
```

Enfin, nous allons placer un marqueur pour indiquer avec un peu plus de précision où se situent les coordonnées fournies par le service de géolocalisation. Le constructeur de l'objet `Marker` utilise les coordonnées ainsi que la carte sur laquelle le placer (`oMap` dans notre cas) :

```
new google.maps.Marker({  
    position: oCoords,  
    map: oMap  
});
```

Allez, on assemble tout ça pour voir ce que ça donne :

Centrer la carte sur notre position et placer un marqueur

```
// on initialise notre carte  
oMap = new google.maps.Map(document.getElementById("map_canvas"), {  
    zoom: 19,  
    center: new google.maps.LatLng(48.858565, 2.347198),  
    mapTypeId: google.maps.MapTypeId.ROADMAP  
});  
// on démarre le suivi de position  
navigator.geolocation.watchPosition(successCallback, null,  
                                     {enableHighAccuracy:true});  
function successCallback(position){  
    oCoords = new google.maps.LatLng(position.coords.latitude,  
                                       position.coords.longitude);  
    // on déplace le centre de la carte à notre position  
    oMap.panTo(oCoords);  
    // on place un marqueur à ces mêmes coordonnées  
    new google.maps.Marker({  
        position: oCoords,
```

```
map: oMap  
});  
};
```

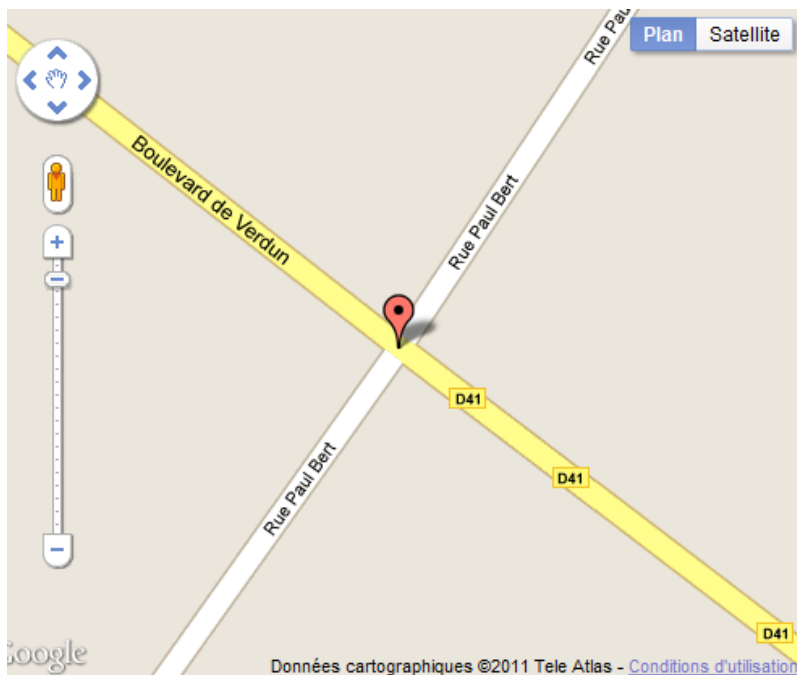


Figure 9.4 — Une belle carte centrée sur le marqueur de notre position !

Bingo ! Ça commence à devenir sympathique ! Mais une seconde... Il y a un petit problème : ce n'est pas du tout ici que j'habite !

9.6.4 Petit point sur la précision

Le service de géolocalisation n'indique pas toujours votre position exacte au mètre près. Si vous ouvrez cette page avec un smartphone équipé d'un GPS en extérieur, alors vous obtiendrez une position très précise. Si vous utilisez une connexion internet mobile de type 3G la position sera un peu moins précise. Enfin si vous utilisez une connexion Wifi ou un réseau filaire en testant par exemple depuis votre ordinateur fixe, le service de géolocalisation utilisera la localisation par IP, qui est très peu précise.

Il se peut également que vous soyez connecté en Wifi depuis un ordinateur fixe mais que vous obteniez une position extrêmement précise, à quelques mètres près. Comment est-ce possible ? Rappelez-vous ! Les Google Cars cartographient les réseaux Wifi ! Par conséquent, cela signifie que l'une des petites voitures de Google est passée dans votre rue et a positionné précisément votre réseau.

Pour ces multiples raisons, il est très difficile de savoir quelle sera la précision des coordonnées récupérées avant de les obtenir. Une fois reçues il est alors possible

d'utiliser la valeur `position.coords.accuracy` afin de savoir si la position est précise ou non. Cette valeur représente le rayon d'incertitude en mètres autour des coordonnées reçues.

Note : les applications Google Maps des smartphones représentent en général cette incertitude par un cercle bleu plus ou moins large selon la précision.

9.6.5 Il est temps de partir en expédition

Bon, revenons-en à nos moutons ! Nous avons un beau marqueur planté au milieu de la carte. Mais si je ne m'abuse nous avons utilisé la méthode `watchPosition()` n'est-ce pas ? Ce qui signifie que si je me déplace dans la rue avec le GPS activé, la carte va suivre mon déplacement ! J'invite les plus enthousiastes d'entre vous à aller tester ceci dans la rue (attention, les passants vous prendront probablement pour un fou qui cherche de l'or enfoui sous le bitume). Pour les autres, je vous assure que cela fonctionne.

De plus, un nouveau marqueur est placé à chaque nouvelle position obtenue. Nous pourrions donc dire que notre application de tracé du parcours est déjà fonctionnelle uniquement avec ces quelques lignes ! Mais comme nous aimons faire les choses bien, nous allons améliorer encore un peu notre application dans cette dernière partie du chapitre de géolocalisation.

9.7 LE TRACÉ (PRÉCIS !) DU PARCOURS

9.7.1 Éviter les bonds de perte du signal GPS

Si vous avez testé notre application en vous promenant autour de chez vous, vous aurez remarqué que lorsque l'on perd le signal GPS (par exemple si l'on passe sous un pont), la carte fait un bond jusqu'aux coordonnées obtenues par géolocalisation 3G. Nous aimerions éviter de traiter ces cas et nous focaliser sur les données précises de type GPS. Pour cela nous allons encadrer le traitement contenu dans notre `successCallback()` par un simple test de précision des coordonnées :

Éviter les bonds de perte de signal GPS

```
function successCallback(position){  
    if (position.coords.accuracy < 50){  
        // Traitement des coordonnées GPS  
    }  
}
```

9.7.2 Savoir quels points relier par un Polyline

Maintenant que nous ne gérons que les données précises, nous allons représenter le déplacement de l'utilisateur par un tracé. Nous utiliserons l'objet `google.maps.Polyline` qui sert à tracer une ligne entre deux points pour dessiner le

parcours effectué sur la carte. Nous avons donc besoin de déclarer une variable qui contiendra en permanence les coordonnées de la position précédente :

```
var oPreviousCoordinates = null;
```

Puis nous traçons la droite entre les deux derniers points à partir de la deuxième position obtenue :

```
// oCurrentCoordinates contient toujours la position actuelle
oCurrentCoordinates = new google.maps.LatLng(position.coords.latitude,
                                              position.coords.longitude);

// gestion du cas du premier traitement
if (oPreviousCoordinates){
    // tracé du Polyline reliant oPreviousCoordinates et oCurrentCoordinates
}
// mise à jour de la position précédente avec la position actuelle
oPreviousCoordinates = oCurrentCoordinates;
```

9.7.3 Tracé du Polyline

Voyons maintenant comment relier nos points à chaque itération grâce à [Polyline](#). Le constructeur d'un Polyline prend en paramètres :

- `path` : un tableau de coordonnées (fournies via des objets [LatLng](#))
- `strokeColor` : la couleur RGB hexadécimale du tracé
- `strokeOpacity` : son opacité
- `strokeWeight` : son épaisseur.

Nous allons dans cet exemple afficher une ligne rouge opaque d'une épaisseur de deux pixels entre nos deux coordonnées :

Tracer une ligne entre nos deux coordonnées avec un Polyline

```
var oNewLine = new google.maps.Polyline({
    path: [oPreviousCoordinates, oCurrentCoordinates],
    strokeColor: "#FF0000",
    strokeOpacity: 1.0,
    strokeWeight: 2
});
```

On ajoute enfin cette ligne à notre carte avec :

```
oNewLine.setMap(oMap);
```

Notre application est maintenant totalement terminée.



Tester l'application et voir l'intégralité du code!
<http://www.livre-html5.com/geolocalisation/map.html>

En résumé

La géolocalisation n'est désormais plus réservée uniquement aux applications natives. Vous avez pu voir qu'en quelques lignes il est très facile d'atteindre des résultats impressionnants sur une simple page web affichée dans le navigateur. On peut donc facilement imaginer de très nombreux cas d'utilisation, comme par exemple aider un utilisateur à remplir un formulaire qui lui demande sa ville en la détectant automatiquement. Si votre utilisation de la géolocalisation est critique, alors prévoyez bien une alternative en géolocalisation par IP si le navigateur de l'utilisateur ne supporte pas celle d'HTML5.

10

Le Web déconnecté

Objectif

Le Web sans connexion Internet, ça laisse songeur. Il s'agit d'une innovation majeure qui rapproche le développement web des applications classiques sur bureau et plus récemment sur mobile.

Nous allons voir dans ce chapitre comment déclarer un site web comme étant une **application *offline***, et quels outils complémentaires utiliser pour **stocker ses données chez le client** avec l'API Web Storage.

Nous allons également détourner cette fonctionnalité pour en faire un mécanisme de super-cache permettant d'accélérer l'affichage de pages plus classiques.

10.1 GÉRER LA DÉCONNEXION

10.1.1 Le cache d'application (AppCache)

C'est quoi une application offline ?

Avant tout il est important de bien comprendre ce qu'est une « application offline ». Ce terme peut facilement être confondu avec les applications qui n'utilisent pas de connexion internet pour fonctionner (par exemple une application de jeu en solo sur un *smartphone*, ou une application de gestion de listes de tâches en local).

Mettons les choses bien au clair : une application « offline » au sens d'HTML5, c'est une page HTML qui reste utilisable lorsque l'utilisateur perd sa connexion internet.

Ce cas arrive fréquemment avec les *smartphones* et autres tablettes en cas de déplacement. Il est d'ailleurs également important de ne pas confondre les applications natives

avec les applications offline HTML5. Une application native aura une apparence et des animations propres à l'OS et sera exécutée sans passer par le navigateur web, tandis qu'une application HTML5 aura l'apparence et le comportement d'une page web classique (rien ne vous empêche de lui donner un look d'application native cela dit !).

Ça marche partout ?

Question compatibilité avec les navigateurs, la problématique des applications offline est assez particulière. Les cas de déconnexions imprévues des ordinateurs de bureau et des ordinateurs portables sont assez rares. Par conséquent le support de la fonctionnalité offline par les navigateurs de ce type d'ordinateurs (Firefox, Chrome, IE, Safari, Opera) n'est pas particulièrement vital. Il l'est en revanche pour les appareils mobiles de type *smartphones*, qui eux, peuvent subir de nombreuses déconnexions (passage sous un tunnel, ascenseur, région à faible connectivité...).

Le support de l'offline par les navigateurs mobiles est donc absolument essentiel. Et voici la bonne nouvelle du jour : iPhone et Android supportent l'offline les doigts dans le nez (depuis iOS 3.2 et Android 2.1 du moins) ! Si vous êtes un fanatique de Blackberry ou Symbian (hahaha...) je vous invite à effectuer vos propres tests pour connaître l'état actuel du support selon les versions qui vous intéressent. En tout cas, avec iPhone et Android, la majorité des smartphones pourra profiter pleinement de l'offline !

Et concernant les navigateurs d'ordinateurs de bureau ? C'est très simple, tous les navigateurs autres qu'Internet Explorer supportent l'offline. Et oui, même IE9 a été un mauvais élève sur ce terrain. Mais peut-être est-ce justement parce que la priorité est aux mobiles ? Dans tous les cas ce détail n'est absolument pas bloquant et il n'est certainement pas une raison suffisante pour passer à côté de la petite merveille que sont les applications offline !

« File-moi ton Cache »

Les applications offline reposent sur l'utilisation de l'AppCache (ou cache d'application). Il s'agit d'un espace supplémentaire géré par le navigateur en plus du cache traditionnel, pour chaque site, (pardon, chaque application web !) afin d'y stocker des fichiers qui seront toujours accessibles même en cas de perte de connexion.

Ainsi, l'AppCache permet de naviguer sur un site sans connexion, d'améliorer la vitesse de chargement des pages, et réduire la charge du serveur. Il amène également la notion de « versions d'applications », concept qui sera détaillé un peu plus loin dans ce chapitre.

Mode hors-ligne ? Cache classique ? AppCache ?

Depuis de nombreuses années déjà, les navigateurs utilisent le cache afin de ne pas re-télécharger certains fichiers statiques (images, fichiers CSS ou JavaScript). De plus, certains navigateurs proposent un mode « hors-ligne » qui permet de se balader entre plusieurs pages en cache. Alors si maintenant il existe un nouveau cache on n'y comprend plus rien !

Tout d'abord le cache classique n'entre pas en conflit avec le nouvel AppCache. Celui-ci va continuer à fonctionner naturellement et indépendamment, sans que nous autres développeurs n'ayons besoin de changer quoi que ce soit.

Concernant la navigation en mode « hors-ligne », il s'agit d'un vestige instable et peu fiable du temps où la bande passante coûtait cher. Si l'on avait besoin de reconsulter la page des horaires d'ouverture d'un magasin, il était économe de pouvoir la retrouver sans connexion. Avec l'avènement du haut débit, cette fonctionnalité est cantonnée à une utilisation par les travailleurs nomades pour signaler à leur navigateur qu'il ne devrait plus aller chercher quoi que ce soit en ligne. Cependant, nous nous retrouvons à nouveau dans un cas similaire avec les connexions mobiles des *smartphones*, qu'il faut ménager le plus possible en raison de leur faible débit et de la couverture réseau instable.

L'apport majeur qu'apporte l'AppCache dans ce contexte est que le développeur a à présent un bien meilleur contrôle sur la gestion du cache de ses utilisateurs, et c'est ce que nous allons voir dans cette partie de chapitre.

10.1.2 Le manifeste d'AppCache

Nous allons maintenant voir comment il est possible de spécifier quels fichiers doivent être mis en cache.

Déclarer le manifeste

Pour utiliser l'AppCache, il faut tout d'abord spécifier l'utilisation d'un manifeste dans lequel seront listés les fichiers à mettre en cache. Ce manifeste se déclare dans votre page HTML.

Nous allons illustrer le fonctionnement de l'AppCache avec cette page « [index.html](#) » dans laquelle nous utilisons une image « [superlogo.png](#) » et un fichier CSS « [style.css](#) » :

Fichier [index.html](#)

```
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href="css/style.css">
</head>
<body>
  <h1>Ceci est mon application offline</h1>
  
</body>
</html>
```

Le manifeste se déclare dans la balise `html` avec l'attribut `manifest`, et par convention, il doit porter l'extension `.appcache`. Appelons donc le nôtre « [manifest.appcache](#) ».

Nous le déclarons dans le fichier HTML de cette manière :

```
<html manifest="manifest.appcache">
```

Un manifeste basique

Il faut bien sûr créer le fichier `manifest.appcache` en question qui est le cœur de la gestion de l'AppCache. Un fichier `manifest` doit obligatoirement commencer par la ligne `CACHE MANIFEST` et peut contenir des commentaires qui sont les lignes commençant par un `#`. On liste dans ce fichier tous les noms des fichiers qui devront être gardés en cache dans le navigateur des utilisateurs. Dans notre cas, il s'agit de nos trois fichiers, `index.html`, `style.css` et `superlogo.png`.

Un manifeste basique

```
CACHE MANIFEST
# Version 0.1
index.html
css/style.css
img/superlogo.png
```

Note : il est possible d'utiliser des URL relatives (comme dans cet exemple) ou absolues (<http://www.une-url.com/absolue/fichier.html>).

Nous verrons un peu plus tard pourquoi les commentaires sont particulièrement importants dans un manifeste d'AppCache.

Les différentes sections du manifeste

Un fichier `appcache` peut contenir plusieurs sections. Chaque section commence par son intitulé (écrit en majuscules et suivi par « : ») puis par la liste des fichiers concernés.

La section `CACHE` (ou la section rien du tout)

Tous les fichiers listés dans la section `CACHE` seront conservés sur le disque de l'utilisateur. Si l'on ne met aucun intitulé de section, nous sommes par défaut dans cette section. C'est le cas dans notre exemple vu un peu plus haut. Nous aurions tout à fait pu l'écrire de cette façon :

```
CACHE MANIFEST
# Version 0.1
CACHE:
index.html
css/style.css
img/superlogo.png
```

Si vous avez peu de contenu dans votre fichier `appcache` vous pouvez vous contenter de ne pas écrire d'intitulé, mais si le fichier commence à devenir volumineux, cela peut améliorer la lisibilité de le mettre.

La section FALLBACK

Cette section est particulièrement importante. Si l'utilisateur tente d'accéder à l'un des fichiers listé dans cette section sans connexion, il sera automatiquement redirigé vers un autre fichier. C'est typiquement très utile pour rediriger vers une page indiquant que l'utilisateur a perdu sa connexion par exemple. Le fichier vers lequel l'utilisateur sera redirigé est à indiquer sur la même ligne de cette façon :

```
FALLBACK:  
/offline.html
```

Note : en utilisant le « / », on indique au navigateur que l'on souhaite que tous les autres fichiers de notre domaine soient redirigés vers [offline.html](#).

La section NETWORK

Vous listerez dans cette section tous les fichiers qui nécessitent une connexion internet. Il peut par exemple s'agir de tous les fichiers qui ne sont pas listés dans les autres sections du manifeste. On peut alors utiliser la notation « * » qui signifie « tout autre fichier n'étant pas dans le cache ». C'est important pour que l'application puisse accéder à des ressources situées sur d'autres domaines que le vôtre :

```
NETWORK:  
*
```

Spécifier le MIME-type du fichier manifest

Enfin, il faut déclarer le type MIME du manifeste avec la valeur [text/cache-manifest](#). Ceci peut être fait via la configuration de votre serveur. Dans le cas d'un serveur Apache pour le PHP, il suffit de créer un fichier [.htaccess](#) contenant la ligne suivante :

```
Les extensions appcache et manifest sont à présent des fichiers manifest  
AddType text/cache-manifest .appcache
```

10.1.3 Un peu de pratique

Structure de projet de test

Afin de bien illustrer le fonctionnement de l'AppCache, voici la structure d'un petit projet de test (figure 10.1).

Notez qu'un fichier [.htaccess](#) invisible sur l'image est également présent à la racine de ce dossier. Il contient simplement la ligne donnée dans la partie précédente.

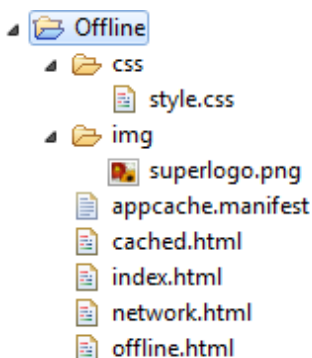


Figure 10.1 — La structure de notre projet.

Voici le contenu du fichier `manifest` :

Manifeste d'exemple

```
CACHE MANIFEST
# v0.1
CACHE:
cached.html
css/style.css
img/superlogo.png
FALLBACK:
/ offline.html
NETWORK:
*
```

Comportement de l'application selon les pages requêtées

Pour tester le fonctionnement de l'AppCache, il faut placer notre projet sur un serveur local (Apache par exemple). Testons à présent le comportement de notre petite application selon les pages requêtées.

Lorsque le serveur Apache est démarré :

- `http://localhost/mauvaiseURL` => Erreur 404
- `http://localhost/index.html` => `index.html` s'affiche, la console indique que l'AppCache est chargé
- `http://localhost/mauvaiseURL` => La page `offline.html` s'affiche
- `http://localhost/cached.html` => `cached.html`
- `http://localhost/network.html` => `network.html`

On remarque surtout que lorsque l'on tente d'accéder à une URL inexistante, si l'AppCache est chargé, alors la page affichée est celle indiquée dans la section `FALLBACK` après le « / ».

Et maintenant si l'on coupe le service Apache :

- <http://localhost/mauvaiseURL> => [offline.html](#)
- <http://localhost/cached.html> => [cached.html](#)
- <http://localhost/network.html> => [offline.html](#)
- <http://localhost/index.html> => [index.html](#)
- <http://localhost/> => [offline.html](#)

Observons la console !

Il n'est pas évident de comprendre ce qui se passe dans le navigateur lorsque l'on utilise l'AppCache. Heureusement pour nous, un petit tour par la console du navigateur nous permettra d'y voir un peu plus clair. Voici ce qui s'affiche lorsque l'on ouvre pour la première fois le fichier [index.html](#) de notre exemple précédent.

Logs affichés par la console lors de la première mise en cache

```
Creating Application Cache with manifest http://localhost/appcache.manifest
Application Cache Checking event
Application Cache Downloading event
Application Cache Progress event (0 of 5) http://localhost/img/superlogo.png
Application Cache Progress event (1 of 5) http://localhost/cached.html
Application Cache Progress event (2 of 5) http://localhost/index.html
Application Cache Progress event (3 of 5) http://localhost/css/style.css
Application Cache Progress event (4 of 5) http://localhost/offline.html
Application Cache Progress event (5 of 5)
Application Cache Cached event
```

Nous voyons ici tous les événements qui ont lieu au fil du remplissage de l'AppCache. Ces événements peuvent être interceptés comme nous le verrons un peu plus loin. Ce qu'il faut bien noter, c'est qu'en ne demandant qu'une seule page l'intégralité des fichiers déclaré dans le manifeste est téléchargée.

10.1.4 Des applications web de plus en plus proches des applications natives

Une version est un « pack » de fichiers

Et que se passe-t-il lorsque le serveur possède une nouvelle version d'un fichier et que l'utilisateur cherche à y accéder (avec une connexion) ? Eh bien cela peut vous sembler surprenant, mais il n'obtiendra pas la nouvelle version ! L'AppCache est considéré comme prioritaire sur la nouvelle version en ligne.

C'est exactement ce qui se passe lorsque vous possédez déjà en cache (classique) un fichier CSS, le navigateur ne télécharge pas à nouveau le fichier, histoire d'économiser un peu de bande passante et d'afficher la page plus vite. Il s'agit d'un comportement plutôt habituel en ce qui concerne les fichiers tels que les CSS, images et JavaScript, mais reconnaissons qu'il est un peu plus perturbant de voir que l'on peut même stocker en cache du contenu HTML.

En fait, les fichiers stockés dans l'AppCache appartiennent à une « version » de votre site. Ils sont donc faits pour fonctionner tous ensemble, et un changement

imprévu dans l'un de ces fichiers ne doit pas casser le comportement stable de l'application. Une application web est en fin de compte très proche d'une application de bureau ou mobile : on télécharge un ensemble de fichiers appartenant à une version, puis de temps en temps une mise à jour de ces fichiers est effectuée pour obtenir la nouvelle version de l'application. On ne veut pas que l'utilisateur obtienne des « inter-versions » avec des fichiers appartenant à une ancienne version et d'autres à la nouvelle. Voilà pourquoi c'est une bonne chose d'avoir des « packs » non scindables de fichiers correspondants à une seule version.

Note : si le téléchargement de l'un des fichiers déclarés échoue, alors l'ensemble du téléchargement échoue, logique !

Mettre à jour l'application web

À présent, comment faire pour faire télécharger à vos utilisateurs la nouvelle version de votre application web ?

Si le moindre caractère change dans votre fichier `manifest`, alors le navigateur tentera de récupérer la nouvelle version de votre application. Et ce caractère peut très bien se trouver dans les commentaires ! C'est là toute l'importance de noter un numéro de version en commentaire afin d'avoir simplement à l'incrémenter pour permettre la mise à jour à vos utilisateurs.

Le navigateur peut vérifier si le manifeste a changé de deux manières :

- Lorsque vous actualisez la page, il lance automatiquement une vérification.
- Si cela lui est explicitement demandé via JavaScript.

Une fois qu'il a détecté un changement, il ne va pas brusquement tout changer dans l'affichage actuel de la page, il attend que l'utilisateur rafraîchisse la page.

Ce qui signifie que si vous ne faites que changer le numéro de version dans votre manifeste, l'utilisateur devra **actualiser deux fois** la page pour voir la nouvelle version de l'application s'afficher. Ce n'est en soi pas gênant pour l'utilisateur, qui n'a probablement pas conscience de l'évolution des versions de votre application, et qui obtiendra des nouvelles versions en naviguant tout simplement sans se poser de question. En revanche en tant que développeur c'est une subtilité qu'il est important de connaître lors de vos tests.

Sachez qu'il est cependant possible d'économiser une actualisation de page en se servant de la Cache API.

10.1.5 La Cache API

Il existe une API JavaScript permettant de manipuler l'AppCache avec un peu plus de précision. Il s'agit de la Cache API, qui permet :

- de connaître l'état actuel du processus de cache,
- d'effectuer des traitements lorsque le processus atteint un nouvel état,
- de lancer automatiquement la procédure d'actualisation du cache.

Tester l'état actuel du processus de cache

L'attribut `window.applicationCache.status` permet de tester l'état actuel du processus de cache, comme dans l'exemple suivant :

```
if (window.applicationCache.status == window.applicationCache.UPDATEREADY){  
    // Traitement  
}
```

Les différentes valeurs constantes que peut prendre cet attribut sont :

- `UNCACHED = 0`, qui est la valeur par défaut lorsque la page n'utilise pas d'App-Cache,
- `IDLE = 1`, lorsque le navigateur possède la dernière version de l'application,
- `CHECKING = 2`, qui est la valeur prise lors de la vérification des mises à jour du manifeste,
- `DOWNLOADING = 3`, lorsque le navigateur télécharge la nouvelle version de l'AppCache,
- `UPDATEREADY = 4`, cette valeur est prise lorsque la nouvelle version est téléchargée, mais pas encore prise en compte,
- `OBSOLETE = 5`, lorsque le manifeste n'a pas pu être trouvé. L'AppCache est alors supprimé.

La plupart du temps, vous devrez probablement plutôt passer par les événements qui sont déclenchés lors des changements d'état.

Les événements des changements d'état

Si vous avez à effectuer un traitement à un instant particulier du processus de mise en cache, voici les événements que vous propose la Cache API :

- `onchecking` : est déclenché quand le navigateur commence à vérifier si des mises à jour sont disponibles.
- `onerror` : est déclenché lorsqu'une erreur se produit (si l'un des fichiers spécifié dans le manifeste est manquant par exemple).
- `onnoupdate` : est déclenché s'il n'y a pas de nouvelle version disponible.
- `ondownloading` : est déclenché au début du téléchargement de l'ensemble des fichiers.
- `onprogress` est déclenché au début du téléchargement de chaque fichier.
- `onupdateready` : est déclenché à la fin du téléchargement de l'ensemble des fichiers.
- `oncached` : est déclenché lorsque le téléchargement est terminé et que le cache est prêt à être utilisé.
- `onobsolete` : est déclenché lorsque le manifeste est introuvable.

Par exemple, si l'on souhaite afficher un message d'erreur particulier si le manifeste est introuvable, nous pouvons le faire de la façon suivante.

Afficher un message d'erreur si le manifeste est introuvable

```
window.applicationCache.onobsolete = function(e) {  
    alert("Erreur de manifest !");  
}
```

Déclencher le processus d'actualisation du cache

L'objet `window.applicationCache` possède également 2 méthodes :

- `update()` : cette méthode permet de déclencher la vérification des mises à jour disponibles sur le serveur. C'est en fait exactement ce qui se produit lorsque vous actualisez la page. Si une nouvelle version est disponible elle est téléchargée, mais l'ancien cache sera toujours utilisé.
- `swapCache()` : cette méthode indique au navigateur qu'il peut commencer à utiliser le nouveau cache. Une actualisation est cependant nécessaire pour afficher la nouvelle version de votre application.

En combinant `update()` qui va régulièrement tester les éventuelles mises à jour de votre AppCache et `swapCache()` pour utiliser la nouvelle version téléchargée, vous économisez donc une actualisation à vos utilisateurs. Le prix est cependant élevé puisqu'à chaque `update()`, une requête est effectuée vers le serveur.

Exemple de déclenchement de l'actualisation du cache

```
// on lance la method update() toutes les minutes  
setInterval( window.applicationCache.update, 60000); // 1 minute  
// lorsque la nouvelle version est téléchargée, on recharge le cache  
window.applicationCache.addEventListener("updateready", function(){  
    window.applicationCache.swapCache();  
}, false);
```

Ce code teste toutes les minutes si une nouvelle version est disponible.

Note : vous pouvez également forcer l'actualisation de la page en JavaScript plutôt que de laisser l'utilisateur le faire manuellement, mais vous prenez alors le risque que ce dernier soit perturbé ou pire, perde ce qu'il était en train de faire sur votre page. C'est donc une pratique déconseillée !

Maintenant que vous maîtrisez parfaitement l'AppCache et les versions de vos applications offline, il est temps de les rendre un peu plus utiles avec l'utilisation du Web Storage !

10.2 STOCKER LES DONNÉES SUR LE CLIENT

Les applications web ont besoin de stocker beaucoup de choses côté navigateur, de l'état de l'application, aux données du serveur visibles par l'utilisateur. Le problème de la persistance de ces informations lorsque le navigateur se ferme puis que la page est

relancée ne se pose pas que pour les applications *offline* : même des sites plus standards peuvent tirer parti de cette fonctionnalité :

- **Longs formulaires sur plusieurs pages** : pour vous éviter de stocker des données incomplètes côté serveur, vous pouvez stocker les données du formulaire chez le client en attendant et recevoir toutes les données en une fois.
- **Volumes importants de données rapatriées** : pour éviter à votre client et à vos serveurs des transits réseau inutiles, et donner à l'utilisateur l'impression de la réactivité, vous pouvez stocker les informations qui ont mis du temps à arriver, comme les points d'intérêt d'une carte, la liste des codes postaux d'un pays...
- **Des images générées sur le client** : une application de manipulation d'images ou même pour consulter hors-ligne un diaporama, vous pouvez stocker des images chez le client en utilisant les `data:uri` et l'encodage en base 64.

Nous allons donc voir deux API de stockage simple (clé / valeur) regroupées dans la spécification **Web Storage**, et nous allons discuter de deux API de plus haut niveau, **IndexedDB** et **Web SQL Database**.

10.2.1 Le stockage simple : Web Storage

Le stockage est étonnant de simplicité mais cette simplicité cache quelques écueils. Commençons par voir le cas simple et sauvegardons quelques données.

Avec ou sans persistance ?

Il existe deux objets répondant à la même API et aux mêmes mécanismes, leur différence ne tenant qu'à la persistance des informations après la fermeture de la page :

- `window.sessionStorage` : les informations restent dans la page et sont détruites à la fermeture du tab ou de la fenêtre.
- `window.localStorage` : les informations survivent aux fermetures de tab et de fenêtre et sont même partagées entre plusieurs tabs ouverts sur le même domaine.

À quoi sert donc `sessionStorage` alors ? Les données ne sont détruites qu'à la fermeture de l'onglet ou du navigateur, vous pouvez donc toujours transporter des informations de page en page sans avoir à utiliser le mécanisme classique *cookie* / stockage serveur.

Les bases

Passons à la pratique et stockons un peu d'information. Dans la plus pure tradition JavaScript, ces objets peuvent être instanciés comme une classe ou accédés comme des tableaux.

Créer / retrouver une donnée, avec des fonctions

```
localStorage.setItem('locale', 'fr-FR');  
console.log( localStorage.getItem('locale') ); // fr-FR
```

Créer / retrouver une donnée comme un tableau

```
localStorage['locale'] = 'fr-FR';  
console.log(localStorage['locale'] ); // fr-FR
```

Vous pouvez même tester directement cette fonctionnalité avec la console JavaScript de votre navigateur :

1. Allez sur n'importe quel site, ouvrez la console JavaScript.
2. Tapez l'une ou l'autre des commandes pour définir une variable.
3. Fermez le navigateur, relancez-le sur le même site.
4. Tapez l'une ou l'autre commande pour retrouver votre donnée.

Effacer les données

```
// supprimer une clé en particulier  
localStorage.removeItem('locale');  
// effacer TOUTES les données du domaine  
localStorage.clear();
```

Attention : les données sont stockées pour chaque nom de domaine complet. C'est bien car cela signifie que *antechrist.example.com* ne peut pas voir ou modifier les données enregistrées par *bisounours.example.com* mais cela signifie aussi que si votre application utilise plusieurs sous-domaines, vous ne pourrez pas utiliser *Web Storage*. À l'inverse des cookies, ceci n'est pas configurable.

Ce que dit l'API : tous en String

L'accès à la fonctionnalité est simple, c'est toujours cela de pris, mais vous allez rapidement vous retrouver à vouloir un peu plus de fonctionnalités. Le stockage clé-valeur c'est amusant pour des démos, mais la plupart des sites auront besoin de stocker des structures un peu plus complexes. Or vous ne pouvez stocker que des chaînes de caractères (*string*). Voici le genre de problème que cela pourrait vous causer.

Vaine tentative de stockage d'un chiffre

```
Var iLastId = 123456 ;  
localStorage['last-id'] = iLastId;  
var iValueOfLastId = localStorage['last-id'];  
(iValueOfLastId === iLastId); // false  
(iValueOfLastId == iLastId); // true  
( typeof iValueOfLastId ) // String
```

Nous avons voulu stocker un entier (*integer*) mais nous retrouvons à la sortie une chaîne (*string*), ce qui peut poser problème si vous codez de manière stricte en vérifiant les types des variables (`===`), ou pire causera des bugs difficiles à trouver si vous codez de manière trop souple (`==`).

Dans ce cas précis, un `parseInt` peut vous tirer d'affaire, mais c'est du code métier que vous rajoutez. Si votre application grossit, vous voudrez certainement éviter d'avoir à taper ce code supplémentaire, et en plus vous voudrez peut-être stocker des tableaux ou des objets complexes. Arrive donc la sérialisation avec JSON.

Sauvegarde de la valeur et du type avec JSON

```
Var iLastId = 123456 ;
localStorage['last-id'] = JSON.stringify( iLastId );
var iValueOfLastId = JSON.parse( localStorage['last-id'] );
(iValueOfLastId === iLastId); // true
( typeof iValueOfLastId ); // integer
```

Avec JSON non seulement vous retrouvez le bon type, mais vous pouvez en plus sauvegarder des données plus structurées.

Sauvegarde de structures complexes

```
Var oLocale = {
  Language : 'fr',
  Country : 'FR'
};
localStorage['locale'] = JSON.stringify( oLocale );
( typeof localStorage['locale'] ) // String
var oStoredLocale = JSON.parse( localStorage['locale'] );
(oStoredLocale.country); // FR
```

Notre API simple prend en fonctionnalité, mais aussi en volume de code à écrire. Nous allons devoir encapsuler tout cela dans une classe façade, d'autant que nous devons aussi gérer les erreurs.

Limites de taille

Lorsque la limite est atteinte, une exception est levée au moment de l'écriture, ce qui peut éventuellement casser l'exécution de votre code. Il vous faut donc utiliser un bloc `try-catch`.

Détecter un débordement

```
try {
  localStorage['language'] = 'fr-FR';
} catch (e) {
  if (e == QUOTA_EXCEEDED_ERR)
    console.log("c'est trop énorme");
}
```

Ce n'est pas esthétique et rajoute énormément de lignes de code, et il paraît vain d'essayer de prédire à l'avance si vous êtes encore en dessous de la limite imposée du navigateur.

Même si l'espace de stockage alloué est beaucoup plus gros que les 4 Ko autorisés des cookies, il y a toujours une limite qui risque d'être atteinte. Elle est de 5 Mo sur Firefox et les Webkit, 10 Mo sur Internet Explorer 8, un pourcentage du disque dur dans certaines conditions sous Chrome et ne parlons pas des *webkits* mobiles dont

les limites varient avec les versions. Si vous y rajoutez les limites des techniques alternatives (64 Ko pour `userData` dans IE6-7 et 100 Ko pour *Flash Stored Object*) ainsi que le fait que d'autres applications du même domaine peuvent utiliser le même espace, autant renoncer immédiatement à essayer de deviner combien d'espace peut être alloué à votre application, et plutôt prévoir dès le départ que vous atteindrez la limite.

10.2.2 Aller plus loin

Stratégie d'utilisation

Aide-toi et le code t'aidera disait la Bible des développeurs : si vous voulez faire quelque chose d'un peu robuste pour gérer vos données tout en présentant une interface simple à votre code métier qui en a bien besoin, nous vous recommandons d'abstraire la gestion de version, les règles de gestion de l'espace et l'encodage / décodage JSON dans une classe façade qui reprendrait l'interface simple de Web Storage : `getItem()`, `setItem()`, `removeItem()`, et c'est tout !

Pour l'encodage/décodage en JSON, reportez-vous à la section « *Ce que dit l'API : tous en String* », vous voyez qu'il y a pas mal de code à prendre en compte, qui peut être mutualisé dans cette classe.

Concernant les règles de gestion des débordements, l'API ne tranche pas sur le sujet, c'est donc à vous de savoir ce que voulez :

- **Règle standard** (*Last In, First Out* ou LIFO) : un nom bien savant pour dire que si la donnée ne rentre pas, tant pis pour elle ; elle n'est simplement pas sauvegardée pour préserver les anciennes valeurs ;
- **Règle du premier arrivé, premier parti** (*First In, First Out* ou FIFO) : tout nouvel ajout de valeur qui provoque un débordement supprime les valeurs des clés les plus anciennes. Cela signifie que vous devez tenir à jour une liste des dates d'écriture de vos clés ;
- **Gestion des priorités** : certaines clés volumineuses (une liste de modèles par exemple) peuvent être sacrifiées pour laisser la place à d'autres fonctionnalités plus importantes (l'auto-sauvegarde de textes en cours d'écriture par exemple) ; si vous les connaissez, vous pouvez donc supprimer les clés les moins importantes lorsque des clés importantes n'arrivent pas à s'enregistrer
- **Date d'expiration** : peut être que certaines de vos données ont de toute façon une durée de vie limitée, donc au moment des débordements vous pouvez les supprimer pour libérer de l'espace.

Vous pouvez choisir de ne pas choisir, auquel cas la règle standard va s'appliquer ce qui équivaut à laisser s'accumuler les premières clés, qui ne sont peut être plus valables ou rarement utilisées.

Enfin nous vous recommandons de gérer les versions de votre site : si vous enregistrez une donnée dans un certain format dans la version 1.1 de votre site, la version 1.2 que vous aurez mis en ligne un mois plus tard devra gérer des clients qui arrivent avec des données en cache à l'ancien format. La méthode la plus simple est

d'enregistrer une clé avec le numéro de version du site, de la comparer au numéro de version actuel, et d'effacer la mémoire si elles sont différentes

Gérer les versions

```
// version actuelle du site
var sVersion = "1.1.1";
// il semblerait que la version ait changé
if( sVersion !== localStorage['version']) {
    // on remet tout à zéro, par sécurité
    localStorage.clear();
}
// et bien sûr on enregistre toujours la version en cours
localStorage['version'] = sVersion ;
```

Cette classe vous permettrait qui plus est d'utiliser de manière transparente pour votre application des méthodes alternatives lorsque le navigateur ne supporte pas Web Storage.

Une note sur la performance

La particularité de `localStorage` est que dès que vous lisez ou écrivez quelque chose, la valeur est retrouvée ou stockée directement sur le disque dur. C'est très pratique car si un navigateur crash brusquement, vos données écrites auront tout de même eu le temps d'être sauvegardées. Par contre cela signifie aussi que si vous lisez/écrivez un grand nombre de fois ou que le disque de votre utilisateur est occupé ailleurs (un scan d'antivirus, une copie...), votre navigateur va freezer le temps que JavaScript finisse d'écrire sur le disque.

Pour éviter cette situation nous vous conseillons là encore de passer par une couche d'abstraction de `localStorage` qui lirait les données depuis la RAM (un tableau JavaScript en fait) et qui écrirait de manière asynchrone et non bloquante (avec la vieille ruse JavaScript qui fait utilisation de `setTimeout`). Voici un exemple d'implémentation.

Lire / écrire de manière non bloquante

```
var myStorage = {
    _keys:{},
    getItem:function(key) {
        // on a peut-être déjà caché cette valeur
        if( typeof(myStorage._keys[key]) !== 'undefined')
            return myStorage._keys[key];
        // récupération depuis le disque et mise en cache
        return myStorage._keys[key] = localStorage.getItem(key);
    },
    setItem:function(key, value) {
        // mise en cache immédiate
        myStorage._keys[key] = value;
        // écriture disque différée
        setTimeout( function() {
            localStorage.setItem(key, value);
        }, 0);
    }
};
```

Cette implémentation est basique mais marche déjà très bien, à vous ensuite d'y rajouter toutes les fonctionnalités dont nous avons parlé précédemment.

On faisait comment avant ?

Croyez-le ou pas, IE6-7 disposait déjà d'un mécanisme de sauvegarde locale des données. Cela s'appelait *userData* et vous pouviez y stocker jusqu'à 64 Ko de données, ce qui est déjà 16 fois plus qu'un cookie, sans les problèmes de performance associés. Firefox 2 et 3.0 avaient inventé *globalStorage*, et on peut toujours compter sur *Flash Shared Object* pour stocker 100 Ko de données (extensible à l'infini si vous demandez la permission à l'utilisateur).

Par ailleurs, si IE8 a bien implémenté une spécification plus ancienne de Web Storage, celle-ci est asynchrone, c'est-à-dire que vous devez passer par une fonction de rappel avant de pouvoir obtenir votre première donnée. Pour faire passer la pilule, vous avez droit à 10 Mo de stockage au lieu de 5 Mo.

Résumons-nous : entre Flash et IE8, nous avons deux méthodes asynchrones, sur des technologies complètement différentes, tandis que IE6-7 et Firefox 2 et 3.0 ont des API dont vous ignoriez jusqu'au nom. Si à ce stade vous avez encore toute votre tête, vous vous demandez comment nous avons osé introduire Web Storage dans ce bouquin censé vous aider à mettre tout cela en production.

Pas de panique, des gens ont travaillé pour vous à résoudre ces problèmes et à les simplifier. Moyennant un passage en asynchrone, vous pouvez utiliser plusieurs bibliothèques très fiables (testées et approuvées par votre serveur en production depuis plusieurs années), que vous devriez rajouter à classe façade que nous vous suggérons tantôt :

- *jStorage* (www.jstorage.info), si votre framework de choix est *jQuery*
- *YUI2 Storage*, qui a choisi Flash, si votre framework de choix est *YUI2*
- *YUI3 Storage Lite*, si vous n'avez aucun de ces deux framework ou si vous utilisez *YUI3*

Note : vous n'aimez pas l'asynchrone ou l'idée de rajouter des bibliothèques ? Il vous reste la stratégie du *progressive enhancement* : développez d'abord sans persistance des données (ce qui est de toute manière recommandé) puis améliorez les choses en n'utilisant la persistance que pour les navigateurs qui supportent le standard *Web Storage*.

Faire communiquer des pages

On peut détourner l'usage original de *localStorage* pour détecter qu'un utilisateur a ouvert plusieurs fenêtres ou onglets du même site, et transmettre des informations entre ces pages (ou dans des *iframes*). Cela peut être utile pour synchroniser des informations comme l'état d'un panier (payé ou non, nombre d'articles) ou savoir quels messages ont été lus ailleurs. Certains utilisateurs aguerris n'hésitent en effet plus à ouvrir plusieurs pages d'un même site pour accélérer leur navigation, pendant

que d'autres laissent ouverts des dizaines d'onglets en permanence, oubliant parfois que votre site est déjà ouvert ailleurs.

Cela se fait simplement en écoutant l'événement `storage` de l'objet `window` et en comparant la valeur modifiée à la valeur dans la page.

```
// nombre d'articles dans le panier pour cette instance de la page
var inNbItems = 3 ;
// écoute de la modification des valeurs stockées
window.addEventListener('storage', function( e ) {
    // quelle valeur a été changée ?
    if( e.key === 'inNbItems' ) {
        if( e.newvalue !== inNbItems ) {
            // la valeur a été mise à jour par une autre page ou frame
            // on synchronise l'interface
            updateNbItems( e.newvalue );
        }
    }
}, false);
```

10.2.3 Le stockage avancé : indexedDB

Dans un souci d'exhaustivité nous allons parler de deux spécifications bien plus complexes que le stockage clé / valeur proposé par Web Storage. Nous allons en parler pour mieux les écarter car vous allez vite voir qu'elles sortent légèrement du cadre de ce livre.

Feu Web SQL Database

Web SQL Database était assez géniale car elle permettait d'avoir une base de données chez le client, à laquelle on accédait en tapant des requêtes SQL, langage généralement bien connu des développeurs web. En théorie cela permettait donc en plus de la persistance des informations d'avoir de quoi filtrer et trier de larges volumes de données, et tous les navigateurs à base de Webkit (Chrome, Safari, Webkit iOS, Webkit Android) ainsi que le navigateur Opéra (bureau et mobile) supportent cette fonctionnalité.

Puissance et large diffusion, qu'est ce qu'on attend pour l'utiliser ? Hé bien elle est maintenant **déclarée obsolète** !

Le W3C s'est retrouvé avec une fonctionnalité beaucoup trop puissante qui demandait à spécifier le langage SQL, ce qui n'est tout simplement pas dans ses attributions. D'un autre côté le WHATWG avec Mozilla, Microsoft et Google a proposé une API sans SQL visible, qui correspond beaucoup plus aux habitudes des développeurs et à la mission du W3C et des navigateurs : *IndexedDB*.

Ajoutez à cet abandon programmé le fait qu'il est pratiquement impossible de reproduire la même fonctionnalité sur les navigateurs sans support (vous vous voyez écrire un interpréteur de SQL en JavaScript ou en ActionScript ?), et vous

comprendrez que cette fonctionnalité géniale n'était que rarement utilisable en production, et donc sortait du cadre de ce livre.

IndexedDB

Son nom officiel est Indexed Database API, mais nous utiliserons son petit nom. C'est donc la survivante du duel avec Web SQL. Elle est de plus haut niveau puisqu'elle permet de manipuler et de stocker directement des objets JavaScript, indexés par clé primaire (auto-incrémentale ou non), le navigateur se chargeant lui-même de la manière dont sont stockées et retrouvées les informations. L'API ne ressemble pas du tout à du SQL mais plutôt à des dizaines d'appels de méthodes pour configurer la base et manipuler les données. Cela dérouterait sans doute la plupart des développeurs web et il faut bien dire que l'API est un peu lourde à digérer mais ça n'est jamais qu'une nouvelle habitude à prendre. Point gênant : on demande son avis à l'utilisateur pour savoir s'il autorise ou non à stocker des informations sur son poste, cela signifie concrètement que votre application doit prévoir le cas où vous n'avez le droit de rien stocker.

Étant donné la jeunesse de cette spécification et de ses implémentations (Chrome et Firefox, IE10 l'ayant mis dans un circuit de laboratoire), et sachant qu'il sera compliqué d'émuler la spécification finale sur tous les navigateurs, nous avons préféré ne pas approfondir le sujet dans ce livre car vous n'êtes pas prêt de vous en servir, même sur des environnements à la pointe comme le mobile.

En résumé

Grâce à l'offline, les applications web mobiles gagnent énormément en flexibilité vis-à-vis des déconnexions. La version web mobile de Gmail en est un parfait exemple. Sans même se rendre compte qu'il a perdu sa connexion, l'utilisateur peut toujours envoyer des mails (ou du moins en avoir l'impression !) car ceux-ci seront mis en file d'attente jusqu'à ce qu'il retrouve une connexion. Pas mal non ?

Si vous n'avez pas tellement la fibre innovante, vous pouvez toujours profiter de l'AppCache et du WebStorage pour améliorer l'expérience utilisateur en leur évitant des allers-retours serveurs ou en sauvegardant leurs informations au fil du remplissage d'un long formulaire par exemple !

Il va falloir attendre un peu ou être très aventurier avant d'utiliser IndexedDB pour du stockage client évolué.

11

WebSockets : l'ère du temps réel

Objectif

Tout s'accélère ma bonne dame, et nos sites web également ! Les applications de type chat, jeux multi-joueurs ou de fil d'infos mis à jour en continu deviennent communes pendant que les requêtes HTTP traditionnelles montrent leurs limites face à ce flot d'information et à la réactivité nécessaire.

Arrive donc un nouveau standard : WebSocket, qui facilite la communication et soulage les serveurs.

Dans ce chapitre nous allons nous initier à WebSocket qui présente l'avantage d'être très simple d'utilisation en créant un mini-chat très réactif qui ouvrira la voie à d'autres applications plus complexes. Nous allons aussi voir les avantages pour vos utilisateurs et votre infrastructure et fidèles à notre habitude, discuter des moyens d'implémenter tout cela en production.

Enfin en complément nous parlerons de Server Sent Event, qui sera plus compatible et mieux adapté à certaines applications.

11.1 L'API WEBSOCKET

11.1.1 Envoyer / recevoir

Il y a des petits bonheurs simples sur Terre comme cueillir des coquelicots frais dans les champs au printemps ou l'arrivée dans les standards d'une API facile à utiliser.

WebSocket est assez bas niveau, aussi établir une connexion permanente avec le serveur et écouter ce qu'il a à dire tient en peu de lignes de code :

Scénario classique d'écoute avec WebSocket

```
var oConnection = new WebSocket('ws://example.com');
oConnection.onmessage = function( e ) {
    console.log( e.data );
};
```

Ici on a ouvert une nouvelle connexion en créant une instance de `WebSocket` (attention à la casse) et en lui donnant l'adresse de notre serveur. Vous noterez le protocole n'est pas `http` mais `ws`. Nous verrons dans la partie serveur ce que cela implique.

Ensuite on se branche sur l'événement `message` de l'objet. La chaîne de caractère envoyée par le serveur se trouve dans l'attribut d'événement `data`. De la même manière que vous utilisez aujourd'hui `XMLHttpRequest`, vous pouvez y mettre ce que vous voulez, et vous finirez probablement par utiliser une chaîne décodable en JSON.

Ce code suffirait déjà de base à la création d'un *widget* d'affichage de cours boursier ou pour surveiller un flux Twitter. Une application plus commune serait d'afficher des résultats d'une recherche qui met du temps à se terminer, comme les recherches de billets d'avion : les premiers résultats peuvent tomber dès la première seconde car ils sont en cache sur le serveur, mais certains prix ne seront pas calculés avant une bonne minute, le temps pour le serveur d'obtenir une réponse de tous les opérateurs ! Utiliser WebSocket permet d'afficher les résultats en temps réel plutôt que d'attendre la fin de la recherche complète.

Puisque le serveur nous parle, répondons-lui ! WebSocket est bidirectionnel et le client a probablement des choses à dire au serveur. Cela se fait simplement avec la méthode `send()` qui prend en paramètre une chaîne de caractère.

Renvoi d'informations au serveur

```
var oConnection = new WebSocket('ws://example.com');
oConnection.onmessage = function( e ) {
    if(e.data === 'PING ?')
        oConnection.send( 'PONG !' );
};
```

Le fait de pouvoir envoyer des informations ouvre la voie aux applications de type jeux, chats, synchronisation de données entre clients...

11.1.2 Gérer la connexion

Toute application connectée qui se respecte a besoin de gérer les erreurs, les déconnexions, les requêtes concurrentes et autres joyeusetés, ce que vous implémenterez probablement dans une classe de plus haut niveau. Terminons donc (déjà) ce tour d'horizon de l'API avec les événements et propriétés qui vous aideront à gérer votre connexion.

D'abord le statut de connexion : il est dans une propriété de l'objet instancié qui s'appelle `readyState` et qui peut prendre les valeurs de quatre constantes :

- `WebSocket.CLOSED`, par défaut (valeur : 3).
- `WebSocket.CONNECTING` (valeur : 0).
- `WebSocket.OPEN` lorsque la connexion est établie (valeur : 1).
- `WebSocket.CLOSING`, selon les implémentations (valeur : 2).

Ensuite les événements `open` et `close` permettent d'être notifié des changements de l'état de connexion, pour programmer par exemple une reconnexion automatique.

Événements importants et utilisation des statuts

```
var oConnection = new WebSocket('ws://example.com');
console.log(
  oConnection.readyState === WebSocket.CONNECTING
); // affiche true
oConnection.onopen = function( e ) {
  console.log( this.readyState === WebSocket.OPEN ); // affiche true
};
oConnection.onclose = function( e ) {
  console.log( this.readyState === WebSocket.CLOSED ); // affiche true
};
oConnection.onerror = function( e ) {
  console.log( e ); // affiche le détail de l'erreur
};
```

11.1.3 Jouons maintenant

Nous allons mettre cela en pratique pour démontrer la réactivité qu'apporte WebSocket à votre application, en construisant un mini-chat. En fait de chat, nous allons synchroniser deux clients (un navigateur de bureau et un *smartphone* par exemple), et chaque lettre tapée d'un côté sera immédiatement affichée de l'autre, ce qui diffère des chats traditionnels où il faut attendre que son interlocuteur ait appuyé sur *Entrée* avant de pouvoir le lire.

Commençons simplement par le HTML où vont s'afficher nos textes

Markup

```
<label>Vous :
  <input type="text" id="parler_ici" placeholder="Quoi de neuf ?" >
</label>
<label>Eux :
  <input type="text" id="ecouter_la" disabled value="que disent-ils ?">
</label>
```

Je ne vous sens pas impressionné du tout. Notez juste les **id** de chaque champ texte pour les avoir en mémoire. Passons au JavaScript : nous allons nous connecter au serveur et l'écouter. Lorsque nous verrons un message commençant par « typing : », nous l'afficherons dans le second `input`.

Connexion et routage du message

```
var oConnection = new WebSocket('ws://example.com');
oConnection.onmessage = function( e ) {
    // expression régulière : on cherche ce qui commence par typing:
    var oRegTyping = /^typing:(.+)/;
    if( oRegTyping.test(e.data) ) {
        // extraction du message et affichage
        document.getElementById('ecouter_la').value
            = oRegTyping.exec( e.data )[1];
    }
};
```

Nous savons maintenant traiter les messages qui viennent du serveur. Il faut supposer que dans le cas de notre exemple nous avons créé un serveur dit de *broadcast* qui se contente de répéter à tous les clients connectés ce que chaque client lui envoie.

Note de sécurité : dans un serveur de *broadcast* simple, toute la partie intelligente du code de notre exemple se trouve côté client, ce qui est un trou de sécurité en soi. C'est suffisant pour une démo, mais bien sûr en environnement de production vous ferez vos vérifications de sécurité côté serveur.

Terminons en envoyant au serveur, et donc par ricochet à tous les clients connectés, le contenu de notre champ texte au fur et à mesure qu'on le tape.

Envoi de notre contenu

```
// à chaque nouvelle lettre, on broadcast le contenu
document.getElementById('parler_ici').onkeyup = function() {
    oConnection.send( 'typing:'+ this.value );
};
```



Voir la démo !

<http://www.livre-html5.com/websockets/minichat.php>

Allez sur cette URL avec au moins votre *smartphone* et votre navigateur de bureau, et notez le **temps de réaction** entre le moment où vous appuyez sur votre clavier virtuel et le moment où elle apparaît sur l'autre navigateur. D'après nos tests sur un mauvais réseau Edge, le délai est en dessous de la demi-seconde, et n'est plus perceptible sur des lignes plus stables.

Personnellement je m'en sers pour copier-coller des URL, des numéros de téléphone ou toute autre chaîne de caractère entre différentes machines ou entre mon poste principal et mon mobile. Il existe des applications mobiles payantes pour ce genre de service, ici une page web suffit...

Remarque : le code de l'exemple que vous trouverez en ligne est à peu près le même que dans ce livre sauf pour l'instanciation de l'objet WebSocket : plutôt que d'appeler directement la classe native du navigateur, nous instancions une classe façade. Cette façade présente la même interface que l'objet WebSocket (méthodes, propriétés, constantes, comportement) ce qui permet de changer facilement l'implémentation sous-jacente et de masquer une complexité trop spécifique qui nuirait à l'apprentissage. En effet, ne disposant pas d'un serveur WebSocket dédié, nous devons emprunter les serveurs de services externes qui demandent tout un protocole d'authentification et l'utilisation d'une librairie dédiée. Ce code étant superflu et volatile, nous avons préféré le cacher.

Vous avez également accès à une seconde démo plus complète qui permet à plusieurs personnes de regarder le même diaporama de photos trouvées sur Flickr, et de regarder en même temps la même photo.



Voir la démo « diaporama collaboratif » !
<http://www.livre-html5.com/websockets/slideshow.php>

11.1.4 Étendre les fonctionnalités

L'API WebSocket est assez bas niveau : vous envoyez et recevez du texte, vous disposez d'événements et de statuts de connexion, et c'est à peu près tout. Vous aurez remarqué dans nos exemples que la fonction centrale est celle rattachée à `onmessage` : c'est un bon point de départ pour router vers votre application métier les différents messages envoyés par le serveur. Les textes échangés étant libres, à vous de définir un protocole de communication.

Dans nos exemples nous préfixons nos messages simplement avec une chaîne définissant le type de message et le caractère « : », suivi de la donnée elle-même (ex. : `typing:tout mon message`). Nous utilisons une expression régulière permettant à la fois de repérer le type de message et d'extraire la chaîne qui nous intéresse.

Selon le volume de données à échanger et les performances, vous voudrez peut-être changer pour des formats plus évolués comme JSON pour gagner en souplesse.

Vous voudrez probablement aussi que la communication avec le serveur soit authentifiée et vous aurez certainement un protocole de négociation de session, pour servir les bonnes données à vos utilisateurs, ou pour vous protéger de demandes illégitimes.

Il vous faut aussi prévoir les cas de déconnexion intempestive et tenter une reconnexion automatique tout en gardant en *buffer* les données que l'application

métier tentait d'envoyer, et il peut être intelligent d'envoyer automatiquement un message de temps en temps au serveur pour ne pas qu'un élément réseau coupe une connexion qu'il pourrait juger inactive (technique dite du « *keep-alive* »).

Enfin WebSocket n'étant pas implémenté nativement sur tous les navigateurs, vous userez probablement de bibliothèques qui font le travail d'utiliser la bonne technologie selon le client.

En bref : construisez autour de WebSocket une couche d'abstraction de tous ces problèmes et ne présentez qu'une interface simple à votre application métier, elle vous le rendra.

11.2 BÉNÉFICES, LIMITES

La communication client-serveur sur le Web se fait en HTTP. J'espère que je ne vous ai rien appris, par contre même les plus chevronnés d'entre nous ont tendance à oublier ce que ce protocole implique en terme de **volume échangé** et de **latence**.

11.2.1 Les performances

Vous saviez qu'un octet et qu'une milliseconde ont un prix ? La performance est la raison d'être de WebSocket, étudions en détail comment cela marche.

Le poids

Voici ce qui passe sur le réseau entre un navigateur et un serveur qui n'aurait pas été spécialement optimisé.

Requête HTTP client vers serveur

```
Host braincracking.org
User-Agent Mozilla/5.0 (Windows; U; Windows NT 5.1; fr; rv:1.9.2.14)
Gecko/20110218 Firefox/3.6.14 GTB7.1 ( .NET CLR 3.5.30729)
Accept text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language fr,fr-fr;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding gzip,deflate
Accept-Charset ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive 115
Connection keep-alive
Cookie utma=45835639.851052137.1281816860.1281816860.1299226320.2;
__utmb=45835639.2.10.1299226320; __utmc=45835639;
__utmz=45835639.1299226320.2.1.utmcsr=(direct)|utmccn=(direct)|utmcmd=(none)
```

Réponse HTTP du serveur au client

```
Date Fri, 04 Mar 2011 08:10:38 GMT
Server Apache/2.2.14 (Unix)
X-Powered-By PHP/5.2.5
Cache-Control no-cache, must-revalidate, max-age=0
Pragma no-cache
```

```
ExpiresWed, 11 Jan 1984 05:00:00 GMT
Content-Encoding gzip
Last-Modified Thu, 03 Mar 2011 09:41:38 GMT
Content-Type text/html;charset=UTF-8
Content-Language fr
Keep-Alive timeout=2, max=90
Connection Keep-Alive
Transfer-Encoding chunked
```

Vous ne maîtrisez pas ce qu'envoie le navigateur. Un cookie peut contenir jusqu'à 4 Ko à lui tout seul, l'en-tête `User-Agent` de certaines versions d'Internet Explorer peut contenir plusieurs kilo-octets (d'ailleurs une des améliorations de performance de IE9 a consisté à avoir une chaîne `User-Agent` fixe et courte). Les autres champs sont classiques, plutôt fixes en taille et en tout cas inamovibles.

Côté serveur, les serveurs les mieux optimisés comme (google.com par exemple) peuvent au mieux diviser par deux le poids de l'en-tête.

L'un dans l'autre, un peu moins de 1 Ko transite entre ce client et ce serveur, et sur certains sites qui maîtrisent mal leurs cookies, on peut monter jusqu'à 8 Ko (aller-retour). Si votre application est un jeu en temps réel, synchronise des clients comme notre diaporama ou va chercher le cours de la bourse, le volume de donnée utile (coordonnées, index, chiffres...) envoyé et rapatrié peut être 10 fois moins important que l'information nécessaire à **chaque** transaction HTTP.

Et alors ? « *En France nous avons des débits parmi les plus rapides du monde* » me direz-vous ? C'est vrai, mais le protocole TCP sur lequel se base HTTP fait que les petits volumes (sous 8 Ko) dépendent surtout de la latence, et non du débit. Concrètement avec une latence de 50 ms entre le client et le serveur, votre requête a voyagé à moins de 800 Ko/s, soit 10 ou 20 fois moins rapidement que ce que les fournisseurs d'accès vendent en ADSL classique. C'est une des raisons qui fait que les sites d'aujourd'hui ne s'affichent pas plus vite que ceux d'il y a 10 ans, malgré des débits multipliés par 1 000 et les optimisations des navigateurs.

Voyez également avec ceux qui s'occupent de l'hébergement du site combien coûtent ces dizaines de kilo-octet inutiles multipliés par le nombre de vos utilisateurs : selon la taille de votre site et votre contrat hébergeur, vous seriez surpris de voir leur importance financière.

Enfin n'oublions ni les mobiles ni les pays qui n'ont que rarement accès à du haut débit : le nombre d'octets qui transitent a un réel impact sur l'expérience utilisateur.

WebSocket utilise le même volume de données que HTTP pour se connecter au serveur, mais il ne le fait qu'une seule fois ! Lorsqu'une donnée utile doit transiter, le protocole ne les accompagne qu'avec 2 octets, au lieu des quelques kilo-octets habituels...

En bref : WebSocket remplace avantageusement les requêtes classiques en divisant par au moins 50 le nombre d'octets accompagnant chaque message.

La latence

La latence est le temps de transit réseau d'un aller-retour client-serveur. Peu de chiffres sont disponibles, mais il semble être de 50 ms en France, oscillant entre 20 ms les jours de pleine lune et plus de 300 ms sur un réseau mobile 3G performant ou un mauvais WiFi. Pour donner un point de comparaison, le cerveau humain commence à percevoir un délai lorsqu'une action prend plus de 150 ms, et les joueurs de *quake-like* savent bien que lorsque leur latence au serveur dépasse ce temps, ils perdent au score.

Traditionnellement aller chercher des informations en continu sur le serveur se fait avec du « *polling* » ou mieux du « *long polling* » : dans les deux cas, à chaque fois que le serveur renvoie une information, le client doit recréer une requête HTTP pour aller vérifier si de nouvelles informations sont déjà disponibles. Si le serveur a des informations disponibles immédiatement, alors vous avez retardé l'arrivée de cette info du temps de latence de la nouvelle requête. Si la latence est de 20 ms, la différence ne sera pas perceptible, mais avez-vous envie de parier là-dessus ?

En bref : Websocket permet de diviser par deux le temps de mise à jour de l'information lorsque celle-ci est mise à jour très fréquemment.

11.2.2 Connectivité

Cross-domain

Contrairement à [XMLHttpRequest](#), il n'y a pas de restriction côté navigateur qui empêche d'appeler un service WebSocket situé sur un autre nom de domaine. Vous pouvez donc fournir un service accessible depuis n'importe où sans avoir à bricoler de solution de type JSON-P (solution classique de contournement de cette restriction, utilisée par la plupart des API du Web). Vous pouvez donc distribuer facilement votre application comme un *widget* d'affichage ou un jeu et continuer à maîtriser la partie serveur.

L'en-tête [Origin](#) est également envoyé à la première connexion et contient le nom de domaine d'où la requête est lancée, vous pouvez donc contrôler et interdire au besoin les accès à votre application. Un pirate qui veut vraiment accéder à votre service n'aura pas de difficultés à modifier ce *header*, mais un site ne pourra pas faire bénéficier de votre application à ses utilisateurs sans votre consentement.

Cookies

Concernant les *cookies*, même lorsque l'appel se fait depuis un autre domaine, ceux-ci sont envoyés au serveur de manière classique à la première connexion, à condition que le navigateur ait déjà reçu des cookies du site web (en [http](#) ou en [ws](#)). Ceci n'est pas un risque de sécurité étant donné qu'il n'y a pas d'accès possible côté client au cookie envoyé par le navigateur *via* WebSocket. C'est très pratique par contre pour reconnaître vos utilisateurs qui voyagent sur d'autres sites.

Attention : vous implémenterez probablement une solution WebSocket qui se base sur Flash pour supporter l'ensemble des navigateurs. Flash n'envoie pas du tout

les cookies à la connexion, il vaut donc mieux ne pas compter dessus et fournir vous-même un mécanisme d'authentification.

Proxys et firewall

WebSocket a été pensé pour passer les *proxys* qui laissent passer les requêtes HTTP normales. Il utilise qui plus est les paramètres de *proxy* du navigateur, contrairement aux *Flash Sockets*, qui se retrouvent de fait bloqués. Il n'est cependant pas impossible qu'une minorité de PALC (Proxys A La C.) refusent la connexion.

Vous aurez noté que le port de connexion à un serveur WebSocket est généralement différent du port 80, même si en théorie vous pouvez configurer votre serveur pour que ça ne soit pas le cas. WebSocket est construit au-dessus de HTTP et commence en fait sa connexion au serveur en passant par le port standard (80 ou 443), ce qui fait que la plupart des firewalls et routeurs ne s'y opposent pas. Cependant certains équipements réseau utilisent des méthodes dites de « *deep packet inspection* » qui analyse la trame HTTP pour y trouver des requêtes douteuses. WebSocket risque de tomber dans cette catégorie.

En bref : même si la situation s'améliore par rapport à Flash Socket, il reste encore certains équipements réseau obsolètes, mal configurés ou très agressifs qui peuvent bloquer les connexions WebSocket. Le mécanisme de gestion d'erreur est là pour ces cas, donc si votre application peut se le permettre, repassez sur les méthodes traditionnelles de `XMLHttpRequest` ou à base d'`iframe`.

11.3 CÔTÉ SERVEUR

C'est là qu'est l'os, hélas. Une implémentation d'un serveur WebSocket n'est pas très complexe à faire, et de fait plusieurs librairies existent pour différents langages. Un serveur WebSocket n'est pas différent dans sa structure d'un serveur HTTP classique et est constitué d'un *process* ou démon qui tourne en permanence. Lorsqu'il reçoit les messages des clients soit il se charge lui-même de réagir et de renvoyer des messages, soit il crée d'autres *process* et attend de manière asynchrone qu'ils y répondent. Il surveille également certaines ressources comme une base de données, des fichiers plats ou attend d'être contacté par d'autre *process* ou par des clients pour renvoyer des informations aux clients.

Le problème, c'est que certains langages ou framework sont mauvais à ce jeu de boucle infinie et de programmation événementielle, mais qu'ils constituent déjà tout le code de votre site « normal ». Oui je pense à PHP.

Ce constat fait, vous avez plusieurs stratégies.

11.3.1 La voie royale

Un serveur WebSocket, c'est avant tout l'avènement de la programmation événementielle côté serveur. Un langage comme JavaScript est donc tout désigné pour cette tâche, et le projet JavaScript serveur sur lequel la majorité de la communauté porte son effort actuel est **Node.js**.

On peut l'installer aussi bien sur un poste UNIX du développeur que sur des serveurs HTTP ouverts sur l'extérieur, et il existe déjà plusieurs bibliothèques de serveurs WebSocket dédiés. Ce projet et les nombreuses bibliothèques qui gravitent autour sont encore jeunes mais bénéficient d'une communauté ultra-active, ce qui peut être un avantage comme un inconvénient en terme de ressources d'apprentissage et de maintenabilité.

Si vous commencez un projet neuf et que vous êtes à l'aise avec l'instabilité du code, c'est votre meilleure option en terme de performances client et de ressources serveur consommées. Vous pouvez réutiliser partiellement l'existant si vous utilisez des API internes ou votre base de données.

11.3.2 La voie diplomatique

Si vous avez trop investi dans le langage et l'environnement de votre site actuel, et que l'application dont va dépendre WebSocket dépend beaucoup du code déjà écrit, vous devez essayer de trouver un serveur WebSocket implémenté en **Java, Ruby Python et même en PHP**. Il n'y a pas encore assez de recul aujourd'hui pour pouvoir affirmer que ces bibliothèques sont de qualité ou même quelle charge elles vont demander à vos serveurs pour tenir ouvertes des dizaines ou des milliers de connexions avec un langage qui n'a pas été fait pour cela originellement.

L'investissement en temps et en infrastructure est à évaluer au cas par cas.

11.3.3 La voie du milieu

Il existe une solution hybride que nous avons utilisée en créant les exemples que vous avez lus : des services externes. Celui que nous avons utilisé s'appelle Pusher¹ et était parfait dans le cadre de nos démos :

- Pas d'installation d'un serveur WebSocket à faire pour nous ou pour vos lecteurs.
- Bibliothèque d'abstraction de WebSocket et des méthodes de contournement fournies.
- Code pour un serveur de *broadcast* (communication aveugle client => serveur => client) déjà créé.
- Possibilité d'utiliser du code de n'importe quel environnement serveur déjà existant.

1. <http://pusherapp.com/>

Les trois premiers points étaient intéressants dans le cadre du livre et peuvent convenir à certains environnements de production. C'est à cause du dernier point que nous nous sommes permis de suggérer une solution commerciale dans un livre technique. Nous n'avons pas d'expérience à grande échelle de ce service en particulier donc nous ne le recommandons pas plus qu'un autre, il s'agit simplement de montrer qu'il y a plusieurs solutions possibles, et qu'à tout le moins vous pouvez implémenter vous-même une telle architecture.

Regardez le schéma de flux d'information (figure 11.1) : votre code serveur actuel peut utiliser ce service pour distribuer une information à un grand nombre de clients connectés en WebSocket, ce qui vous donne donc la réactivité et le support d'un grand nombre de connexions.

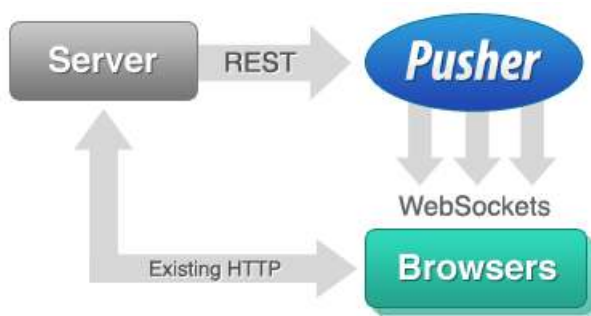


Figure 11.1 — Exemple de flux d'information utilisant un serveur WebSocket et un serveur classique, tel qu'implémenté par le service payant Pusher.

Le client lui continue à dialoguer avec vos serveurs par des requêtes XHR normales, utilisant ainsi le code que vous avez déjà mis en place. Ce schéma n'est pas idéal pour des jeux en temps réel par exemple, mais parfait pour un chat, un jeu massivement distribué au tour par tour, la distribution de cours boursiers à un grand nombre de clients etc.

Dans le même esprit, pour le même type d'applications, vous pouvez utiliser le standard *Server Sent Event* si vous ne voulez pas dépendre d'un service externe.

11.4 ALLER PLUS LOIN

11.4.1 L'API Server Sent Event

Avant WebSocket, *Server Sent Event* a été la première spécification à implémenter dans le navigateur une communication qui reste ouverte sur le serveur et qui permet de recevoir des événements de celui-ci. On n'est plus dans la communication bidirectionnelle car le canal n'est ouvert que du serveur au client, ce qui pourra suffire à la plupart des applications. Cela n'empêche bien sûr pas le client de communiquer avec le serveur avec des XHR classiques.

Quel est l'avantage de cette spécification si elle en fait moitié moins que WebSocket ?

- **Côté serveur** : pas de *Node.js* ou d'implémentation spécifique, vous pouvez réutiliser votre infrastructure actuelle.
- **Côté client** : l'API est plus haut niveau que WebSocket et un certain nombre de fonctionnalités comme la reconnexion automatique et les événements nommés sont gérés par le navigateur.

C'est finalement une standardisation des solutions dites de « *forever iframe* ». Voyons par exemple un code typique, côté serveur, en PHP de ce type de solution.

Code d'envoi de données via une iframe

```
<?php
// on ne veut surtout pas que le navigateur mette en cache cette page
header('Cache-Control: no-cache');
// boucle infinie
while( true ) {
    // méthode métier de récupération de données (base, fichier plat, service)
    $data = get_data() ;
    // appel d'une fonction métier avec les données PHP au format JSON
    print '<script> fonction_rappel(';
    print json_encode( $data );
    print ');</script>';
    // envoi des données dès qu'elles sont disponibles
    flush();
    // on laisse respirer un peu, et on y retourne
    sleep(1);
}
```

Dans le cas d'une réponse à un appel *via Server Sent Event*, le principe de la boucle infinie reste, nous allons apporter quelques modifications légères :

1. Modification de l'en-tête `Content-Type`, ça n'est plus `text/html` mais `text/event-stream`.
2. Plus besoin de balises `script` et d'une fonction de rappel, chaque ligne de donnée est préfixée avec `data:`,
3. Détection du support de SSE du navigateur, grâce à l'en-tête `HTTP_ACCEPT` envoyé par le navigateur.

Code compatible iframe / SSE

```
<?php
// le navigateur nous dit si il supporte le SSE ou pas
$hasSSE = ($_SERVER['HTTP_ACCEPT'] === 'text/event-stream');
// on ne veut surtout pas que le navigateur mette en cache cette page
header('Cache-Control: no-cache');
// envoi de l'en-tête approprié
if($hasSSE)
    header('Content-Type: text/event-stream');
// boucle infinie
while( true ) {
```

```

    // méthode métier de récupération de nouvelles données (base, fichier plat,
service ...
    $data = get_data();
    $data = json_encode( $data );
    if($hasSSE)
        print 'data:'. $data ;
    else
        print '<script> fonction_rappel('. $data . '); </script>';
    // double retour chariot indispensable
    print PHP_EOL.PHP_EOL;
    // envoi des données dès qu'elles sont disponibles
    flush();
    // on laisse respirer un peu, et on y retourne
    sleep(1);
}

```

Pour que les données soient considérées comme faisant partie du même bloc, il faut qu'elles tiennent toutes sur une seule ligne préfixée par `data:`. C'est ce que fait par défaut la fonction PHP `json_encode`. Si vos données doivent comporter des retours chariots, chaque nouvelle ligne doit être également préfixée. Pour séparer deux blocs de données, un double retour chariot est nécessaire.

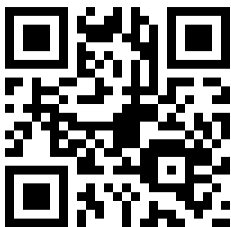
Côté client, pour un navigateur supportant l'API, le code de base est assez simple.

Code client de récupération des données

```

var oConnexion = new EventSource('http://example.com/getdata.php');
oConnexion.onmessage = function( e ) {
    console.log( e.data ); // en sortie : la chaîne JSON
};
oConnexion.onerror = function() {
    console.log('erreur');
}
oConnexion.onopen = function( e ) {
    console.log('connecté'); // en sortie : une chaîne avec le message d'erreur
}

```



Voir une application d'horloge !

<http://www.livre-html5.com/websockets/sse-timer.php>

L'API Server Sent Event est un peu plus haut niveau que WebSocket, et gère toute seule les reconnexion au serveur. Vous pouvez également :

- **Gérer des tickets** : si le serveur renvoie `id:une_chaine`, le client renverra à la prochaine reconnexion la valeur « `une_chaine` » dans l'en-tête `Last-Event-ID`. C'est généralement un marqueur de temps ou un numéro de ligne qui est envoyé, ce qui aide le serveur à renvoyer des données à partir du dernier point.
- **Définir le délai avant reconnexion**, en renvoyant du serveur `retry:` avant le nombre de secondes.

- **Nommer des événements** : si la ligne `data:` est précédée d'une ligne commençant par `event:une_chaine`, alors le client peut se brancher sur ces événements-là.

Arrêtons-nous un instant sur les événements nommés et la gestion de tickets avec un exemple concret dans la même réponse serveur.

Événements nommés : chaîne envoyée par le serveur

```
event: tic-tac
data: Il est 23:45:54
id: 123456789
```

Événements nommés : objet interprété par le client

```
var oSource = new EventSource('http://example.com');
oSource.addEventListener('tic-tac', function( e ) {
    e.type ; // 'tic-tac'
    e.lastEventId ; // '123456789'
    e.data ; // 'Il est 23:45:54'
}, false);
```

En bref : *Server Sent Event* est une alternative intéressante à WebSocket si l'application ne fait que recevoir des données serveur. Elle permet de réutiliser pratiquement le même code serveur que les techniques de *long polling*. Côté client, elle peut être remplacée par la technique des « forever iframe » si le navigateur ne la supporte pas.

11.4.2 On faisait comment avant ?

Voyons les techniques utilisées aujourd'hui par les sites pour faire du quasi-temps réel. Elles sont toutes moins réactives et plus gourmandes en trafic que WebSocket mais présentent l'avantage d'être grandement compatibles et facilement implémentables. Dans tous les cas, **elles ne conviennent que difficilement aux applications où le client envoie très fréquemment des informations**, comme les jeux. Ces trois techniques gagnent à chaque fois en complexité et en efficacité sur la précédente, au développeur de savoir laquelle correspond à son utilisation.

Note : dans les tableaux suivants, chaque nouvelle ligne peut représenter une seconde. La donnée utile est toujours renvoyée à la troisième seconde.

Le « polling »

Basique et facile à implémenter : dans une boucle infinie, le client demande à intervalle régulier au serveur *via* XHR si il a de nouvelles données à lui envoyer.

En contrepartie, très peu de réactivité, et énormément de trafic potentiellement inutile. On a vu que n'importe quelle requête HTTP pouvait prendre entre 1 Ko

Tableau 11.1 — Dialogue client-serveur « *polling* »

Client	Serveur
Ping ?	rien
Ping ?	rien
Ping ?	Pong !
Ping ?	rien

et 8 Ko, imaginez ce que donne une requête par seconde, multipliée par le nombre d'utilisateurs simultanés.

Cette technique suffit pour des mises à jour très peu fréquentes et des sites à petit trafic ou avec des capacités serveur surdimensionnées

Le « *long polling* »

Vous pouvez améliorer la réactivité et baisser le nombre d'octets inutiles échangés en optant pour des XHR qui ne sont pas fermées par le serveur.

Tableau 11.2 — Dialogue client-serveur « *long polling* »

Client	Serveur
Ping ?	
	Pong !
Ping ?	

Celles-ci durent alors jusqu'à ce que le navigateur, le serveur ou un proxy détecte un *timeout*, ou bien lorsque le serveur renvoie une nouvelle donnée. La connexion est alors immédiatement rouverte par le client.

Cette technique est bonne pour une fréquence de mise à jour espacée dans le temps. Le serveur doit être dimensionné pour garder suffisamment de connexions constamment ouvertes.

Les « *forever iframe* »

Pour encore gagner en réactivité dans le cas où vous avez des mises à jour serveur fréquentes, vous pouvez opter pour ce *hack*. Côté client, on crée dynamiquement une *iframe*, côté serveur on remplit cette *iframe* grâce à une boucle infinie qui écrit régulièrement des balises *script* avec ou sans nouvelles données à l'intérieur. C'est au client de détecter la fin de cette connexion HTTP et de recréer alors une autre *iframe* pour continuer la récupération des données.

Par contre, lorsque le client doit envoyer des informations, il doit le faire en XHR, donc sur un autre processus serveur. C'est alors au serveur de gérer les allers-retours et de faire communiquer deux processus différents, par exemple par le biais d'une base de données.

Tableau 11.3 — Dialogue client-serveur « *forever iframe* »

Client	Serveur
Ping ?	rien
	rien
	Pong !
	rien
Timeout !	
Ping ?	rien

Cette technique est à préférer au *long polling* dans le cas de mises à jour serveur fréquentes et lorsque le dialogue du client au serveur n'est pas important. La spécification *Server Sent Event* remplit exactement ce rôle, avec l'avantage d'une formalisation des transmissions et une implémentation native dans certains navigateurs.

11.5 EN PRODUCTION

Nous avons donc deux cas de figure, en fonction du besoin de votre application métier :

- Si vous avez besoin d'une communication bidirectionnelle et d'une grande réactivité, même sur mobile, vous devez rester sur les WebSocket ;
- Si la réactivité peut être sacrifiée et que seul le serveur a des informations à envoyer, *Server Sent Event* et son équivalent en *hack* « *forever iframe* » peuvent suffire. Référez-vous au paragraphe *L'API Server Sent Event*.

Pour WebSocket, nous avons vu dans le paragraphe Côté serveur vos options côté serveur, quelles sont vos options côté client ?

Il n'y en a qu'une de vraiment éprouvée, c'est l'utilisation de *Flash Socket*. La spécification de WebSocket s'en est d'ailleurs inspirée, et la fonctionnalité existe dans le logiciel d'Adobe depuis la version 9, très largement répandue sur les postes clients.

Cela consiste donc à invoquer un fichier `.swf` qui ne contient pas d'interface mais qui se charge de gérer la connexion. La connexion est pilotée depuis JavaScript grâce à un mécanisme d'Adobe nommé *ExternalInterface*.

11.5.1 Contraintes

L'avantage de Flash est qu'il est très largement répandu sur les navigateurs de bureau, mais l'utilisation des WebSocket natifs permet d'éviter les écueils de Flash :

- Les *cookies* ne sont pas envoyés avec la première connexion. Si vous avez besoin de renvoyer des données privées, il vous faudra donc implémenter votre propre mécanisme d'authentification, en envoyant par exemple la valeur du cookie dans le premier message du client au serveur.

- Les *proxys* ne sont pas passés facilement car Flash ne les connaît généralement pas. Cela signifie qu'il vous faudra afficher un message d'erreur pour ces personnes ou prévoir une méthode alternative de *long polling* moins efficace, mais passe-partout.
- Pour faire des requêtes *cross-domain*, il faut installer et maintenir sur le serveur cible un fichier à la racine du site nommé `crossdomain.xml`, qui liste les domaines d'où les appels sont autorisés. Pour ne pas avoir à maintenir cette liste, on peut simplement déclarer autoriser tous les domaines.

11.5.2 Librairies

Le plus efficace est d'utiliser des librairies client qui vous fournissent les fichiers `.swf` et `crossdomain.xml` prêts à l'emploi, ainsi que des instructions de déploiement. Elles se chargent en plus d'utiliser WebSocket natif lorsqu'il est disponible et proposent généralement une API qui imite l'API WebSocket officielle :

- **socket.IO Client**¹ est la plus connue et surtout la plus complète : outre WebSocket et Flash, elle bascule automatiquement dans la technique de *long polling* la plus adaptée au navigateur. Si vous optez pour *Node.js* en serveur, elle s'interface parfaitement avec **Socket.IO-node**².
- **web-socket-js**³ est la plus ancienne et est restée simple, elle se contente d'utiliser le WebSocket natif ou d'utiliser Flash.
- quelques fournisseurs de serveur WebSocket vous proposent leurs librairies. Il en existe pour GWT, jWebSocket ou Jetty (java).
- chaque fournisseur de service WebSocket vous proposera sa librairie, qui présente à chaque fois l'avantage de négocier l'authentification à leurs serveurs pour vous. Citons *Kaazing* et *Pusher App*.

En résumé

Nous avons vu l'efficacité de WebSocket en action, et comment utiliser Flash pour en faire bénéficier tous les navigateurs. Vous savez maintenant à quoi vous en tenir concernant l'implémentation en production avec des librairies, et quelle est la meilleure stratégie à adopter côté serveur. Nous avons vu que *Server Sent Event* et les techniques de *polling* peuvent parfois suffire, malgré les problèmes de performance qu'ils posent aux sites actuels.

Bref choisissez bien votre technique selon le besoin de votre application et vous voilà paré pour construire des applications web plus réactives que jamais !

1. Socket.io, code client : <https://github.com/LearnBoost/Socket.IO-node-client>.

2. Socket.io, code serveur : <https://github.com/LearnBoost/Socket.IO-node>.

3. Code Web-socket-js : <https://github.com/gimite/web-socket-js>.

Conclusion

Nous voilà donc à la dernière section de ce livre. Si tout s'est passé comme prévu vous devriez en savoir un peu (voire beaucoup !) plus sur HTML5. Vous êtes maintenant capables de transformer une simple page web en « application web », et ça, c'est la classe. *Drag and Drop*, géolocalisation, *offline*, WebSockets, multimédia, super-formulaires... vous avez de quoi vous amuser pour les prochaines années.

Au cours de cette conclusion nous allons faire le point sur les API qu'il vous reste à explorer et sur ce qui nous attend pour les années à venir. Nous terminerons par un petit point sur le CSS3 et nous décrirons quelques outils qui vous aideront à mettre en place de vrais projets HTML5 en production.

QUELQUES AUTRES API NON COUVERTES

Sachez qu'HTML5 regorge encore de nouvelles fonctionnalités incroyables qu'il vous reste à découvrir. Sans trop entrer dans les détails, voici quelques pistes de sujets sur lesquels vous pouvez vous documenter sur internet :

- **L'History API et l'événement hashchange** – L'History API permet d'interagir avec l'historique du navigateur de l'utilisateur. Vous pouvez modifier entièrement l'URL actuelle sans rechargement de page, accéder à l'historique ou encore faire retourner le navigateur sur la page précédente par exemple. Actuellement, les applications AJAX mono-page utilisent généralement le # (ou *hash*) pour gérer la navigation. Grâce à HTML5 (et depuis IE8), vous avez accès à l'événement [hashchange](#) qui simplifie la détection des changements de hash. L'History API permettra à terme de s'affranchir de l'utilisation du hash pour gérer la navigation dans ces applications, et de revenir au bon vieux principe du Web : 1 URL = 1 page.
- **L'attribut contenteditable** – Inventé par IE 5, cet attribut HTML transforme un élément classique en élément éditable. Il devient alors possible d'éditer le contenu d'une simple balise [p](#) comme s'il s'agissait d'une [textarea](#). Il peut être alors tout à fait envisageable d'éditer un wiki ou un article de blog en restant sur la page de consultation. Cela ravira les utilisateurs aguerris mais également et

surtout les débutants qui n'ont pas l'habitude d'éditer un contenu sur une page d'administration.

- **WebGL pour la 3D** – Nous avons vu dans ce livre le fonctionnement basique de Canvas. Comme nous l'avons déjà abordé dans le chapitre en question, il est possible d'utiliser WebGL afin de réaliser des graphismes en 3D très poussés. Si vous vous intéressez à la 3D ou aux jeux vidéo, c'est clairement une bibliothèque sur laquelle il va falloir se pencher !
- **Web Workers** – Lorsqu'une page web exécute un code JavaScript, celui-ci bloque le navigateur le temps de son exécution. C'est pour cela que lorsqu'une boucle infinie est exécutée... le navigateur plante. Ou du moins, l'onglet actuel plante. L'idée des Web Workers est de rendre le traitement de certains scripts lourds non bloquants en les exécutant dans un autre thread du système. Les calculs lourds se faisant généralement côté serveur, les cas d'utilisation de cette technologie sont encore trop rares sur des sites web. De plus, émuler cette fonctionnalité est complexe et marche assez mal. Nous n'avons donc pas abordé le sujet.

LE FUTUR DES APPLICATIONS WEB

Lorsqu'on entend parler de HTML5 dans la presse, on en parle surtout au futur et avec des étoiles dans les yeux. Ce bouquin est plus terre à terre et vous a présenté ce qui marche aujourd'hui, et surtout ce qui va servir aux sites actuels pour améliorer l'expérience utilisateur. Prenons tout de même le temps de voir ce qui arrive, car à la vitesse où vont les choses sur certaines plateformes, il n'est pas impossible que vous ayez prochainement à jouer avec ces API même expérimentales :

L'accès au matériel

L'accès au matériel est sans doute le plus gros et le plus prometteur des chantiers du W3C car on pourra mettre en concurrence directe les technologies du Web avec les langages des applications natives.

La géolocalisation (chapitre 9) avait ouvert le bal en transmettant des informations de la puce GPS, Wifi ou de l'antenne à un navigateur puis au code JavaScript.

Avec l'API `getUserMedia` on pourra accéder au microphone ou à la caméra d'un mobile, d'un PC classique ou d'une console d'une manière uniforme et standardisée. L'utilisateur aura alors le choix entre prendre une photo, une vidéo, un son directement ou bien en choisir une dans sa bibliothèque. En couplant cette fonctionnalité avec les Web Workers et Canvas, on peut imaginer des applications de reconnaissance faciale, des jeux qui exploitent les mouvements ou les expressions, etc. Android 3¹, Opéra bureau et Chrome supportent déjà cette API, tandis que le W3C travaille avec Ericsson ou Nokia sur le sujet.

1. <http://davidbcalhoun.com/2011/android-3-0-honeycomb-is-first-to-implement-the-device-api>

Le gyroscope sera accessible avec l'API (<http://dev.w3.org/geo/api/spec-source-orientation.html>) *DeviceOrientation*¹, et vous permettra donc de connaître l'orientation de l'objet. Une implémentation existe déjà sur Chrome et Firefox sur les Mac Book Pro mais étant donné le poids de l'objet, il est difficile d'imaginer une application pratique. Par contre, les mobiles, tablettes et consoles portables permettront d'exploiter à outrance ces API dans le cadre d'applications de jeu par exemple, aussi Android 3, Opera mobile et iOS 4.2 ont tous implémenté l'interface.

Le peer to peer ?

Faire parler deux clients directement sans passer par le serveur, voilà qui ne s'est encore jamais vu dans le Web. Et pourtant des essais d'implémentation d'un protocole² de *peer to peer* (*ConnectionPeer*) ont été menés par Ericsson Labs et permettent de faire un chat vidéo sans abuser du serveur, celui-ci ne servant que pour la transmission des métadonnées à côté de ça, le protocole WebRTC³ prend de l'ampleur avec des expérimentations et des implémentations dans les versions de laboratoire de Firefox et Chrome.

L'accès au bureau

L'API de notification⁴ déjà intégrée à Chrome et utilisée par Gmail permet à une page web d'afficher un message sur le bureau de l'utilisateur même lorsqu'il ne regarde pas la page, comme Growl sur Mac ou les notifications Push d'iOS. La spécification et les implémentations de Firefox et Chrome bougent encore beaucoup.

Vous pourrez demander à l'utilisateur d'accéder à sa liste de contacts grâce à la Contacts API⁵ ce qui convient bien à une utilisation sur un téléphone, ou pour les utilisateurs Mac OS et Windows qui utilisent les programmes de mail par défaut.

Les graphismes

WebGL apporte le 3D dans les pages, et Canvas et SVG deviennent des références concernant la 2D. On pourra objecter que SVG a déjà 10 ans d'âge, mais cette technologie va enfin pouvoir être utilisée de par le support navigateur grandissant. Canvas arrive doucement et fait rêver beaucoup d'éditeurs de jeux vidéo ou de logiciels de manipulation d'images. On ne sait pas encore ce que l'on pourra imaginer comme environnement 3D avec WebGL : à minima des jeux, sinon de nouvelles manières de naviguer.

1. <http://dev.w3.org/geo/api/spec-source-orientation.html>

2. <http://www.whatwg.org/specs/web-apps/current-work/multipage/dnd.html#peer-to-peer-connections>

3. <http://www.webrtc.org/>

4. <http://www.w3.org/TR/notifications/>

5. <http://www.w3.org/TR/contacts-api/>

Soyons fous, rajoutons CSS3 et ses modules expérimentaux d'animation 2D et 3D, la maîtrise de la disposition des pages ou de la génération d'effets graphiques et de contenus qui vont nous permettre une meilleure séparation de la présentation et des données. Microsoft, Adobe, Apple et les autres font des propositions et des implémentations tous azimuts, rarement compatibles mais toujours prometteuses de CSS.

Le multi-terminal

Nous sommes clairement déjà dans un monde où l'utilisateur peut et veut accéder au même service de plusieurs manières : lire les infos sur sa tablette au petit-déjeuner, continuer sur son téléphone dans les transports ou se les faire lire dans sa voiture, poursuivre à la pause de midi sur son navigateur de bureau, et le soir obtenir des informations écrites complémentaires au journal télévisé, directement sur le téléviseur. On n'est plus dans la science-fiction mais déjà dans le présent ou dans un futur très proche, et il est probable qu'il s'agira d'un marché de masse.

Les fournisseurs de contenu ne peuvent pas dépenser autant d'argent pour s'adapter à tous ces médias, aussi un langage universel a une justification économique immédiate. Visiblement la langue de toutes ces interfaces sera le triptyque HTML / CSS / JavaScript.

ET LE CSS3 ALORS ?

Il arrive très souvent que l'HTML5 soit naïvement assimilé au CSS3 et que l'on entende des remarques du style : « Avec HTML5 maintenant on peut faire des bords arrondis sans images ». Il est important de bien séparer ces deux technologies et ne pas faire d'amalgames trop rapides.

Le CSS3 est l'ensemble de nouveautés apportées aux feuilles de style depuis CSS2.1. Ces nouveautés n'ont en soi aucun rapport avec HTML5 puisqu'elles concernent l'apparence et non la sémantique ou le fonctionnel. Il est d'ailleurs bien sûr parfaitement possible de styliser une balise HTML5 avec du CSS2.1 ou une balise HTML4.01 avec du CSS3.

HTML5 et le CSS3 sont simplement deux nouvelles technologies qui font parler d'elles et qui sont implémentées par les navigateurs au même moment. Le CSS3 apporte une palette de nouveaux sélecteurs CSS, d'effets visuels, de nouvelles manières d'écrire les couleurs, et bien d'autres choses encore.

Nous n'évoquerons pas en détail le contenu du CSS3 mais sachez qu'il est de bon augure de vous y intéresser en parallèle à HTML5. Un site en HTML5 est un site qui se veut moderne et qui prend en considération les derniers outils disponibles, dont le CSS3 fait partie. Autant dire que coder son site avec la syntaxe et les balises HTML5 est absolument incompatible avec l'utilisation d'un tableau à neuf cases pour faire des bords arrondis autour d'une `div`...

Pour les petits curieux voici quelques-uns des effets visuels les plus tape-à-l'œil que vous pouvez utiliser en CSS3 :

- des ombres sur les textes ou les blocs,
- des dégradés (linéaires et radiaux),
- des bordures arrondies,
- des polices embarquées dans votre page,
- gérer la transparence d'un élément,
- effectuer des rotations, translations et étirements d'éléments,
- ...en 2D et en 3D,
- ...le tout avec des animations de transition entre le passage d'un état à un autre.

Eh oui, c'est encore tout un monde merveilleux à explorer !

Si vous voulez vous former à ce langage émergent, nous vous recommandons chaudement le livre paru chez Dunod, édité par notre ami et néanmoins confrère Vincent De Oliveira, *CSS3 le design web moderne*.

LES OUTILS POUR HTML5

HTML5shiv



HTML5shiv est hébergé sur Google Code.

Les navigateurs modernes tels que Firefox, Chrome, Safari, Opera et Internet Explorer 9 reconnaissent très bien les nouvelles balises sémantiques HTML5. Cependant les versions d'Internet Explorer 8 et inférieures ont besoin d'un petit coup de pouce afin de styliser ces nouvelles balises, qui par défaut ne sont pas reconnues (elles n'apparaissent même pas dans le DOM). Ce petit coup de pouce se nomme HTML5shiv. Il s'agit d'un petit fichier JavaScript qui permettra à Internet Explorer de pouvoir placer les balises HTML5 dans le DOM et donc de les styliser. Il est basé sur la méthode `createElement` :

La base du « hack »

```
document.createElement("header");
```

Tous les nouveaux éléments sont automatiquement créés avant que le contenu du corps de la page ne soit affiché. Il est ainsi obligatoire d'exécuter HTML5shiv dans le `head` de la page. Une fois ce script exécuté, Internet Explorer 6, 7 et 8 peuvent alors soudainement traiter les balises HTML5 aussi bien qu'ils le font avec nos bons vieux `div` ! De plus, HTML5shiv rend ces éléments imprimables sur Internet Explorer, ce qui n'est pas le cas lorsque `createElement()` est utilisé seul.

Modernizr 2



Logo de Modernizr.

Au cours de vos développements HTML5, il arrivera forcément que vous ayez besoin de savoir si telle ou telle fonctionnalité est supportée par le navigateur afin de proposer une solution alternative. Par exemple voici comment on teste le support de la géolocalisation.

Tester le support d'une fonctionnalité

```
if (navigator.geolocation)
    alert("Géolocalisation supportée");
else
    alert("Géolocalisation non supportée");
```

Mais toutes les fonctionnalités ne peuvent pas être testées si facilement. Et quand bien même elles le seraient, il n'est pas évident de se rappeler de toutes les techniques permettant de les tester. Certaines d'entre elles sont moins intuitives.

Test du support de la Canvas Text API

```
var c = document.createElement('canvas');
return c.getContext && typeof c.getContext('2d').fillText == 'function';
```

Test du support du Drag and Drop

```
return 'draggable' in document.createElement('span');
```

C'est à ce moment-là qu'intervient Modernizr, une petite bibliothèque JavaScript permettant d'abstraire ces tests de support de fonctionnalités. Ils s'utilisent ainsi tous de la manière suivante :

Test par Modernizr

```
if (Modernizr.geolocation){
    alert("Géolocalisation supportée");
else
    alert("Géolocalisation non supportée");}
```

Simple non ?

Notez au passage que certains navigateurs mentent sur leurs capacités et se vantent d'être très avancés dans le support de l'HTML5. Une bibliothèque telle que Modernizr permet de révéler au grand jour quel est leur niveau de support réel.

Modernizr inclut HTML5shiv, et doit par conséquent être impérativement exécuté dans le [head](#) de votre page.

Une fois le code exécuté, vous pouvez accéder via l'objet Modernizr à un ensemble de valeurs booléennes qui vous indiqueront quelles fonctionnalités sont supportées. Cet ensemble de fonctionnalités est listé dans la documentation du site modernizr.com.

D'ailleurs si vous faites un tour sur modernizr.com (qui utilise bien évidemment Modernizr) avec votre navigateur et que vous inspectez le code source (via Firebug par exemple), vous vous rendrez compte qu'un ensemble de classes CSS a également été ajouté à la balise `html` :

Les classes ajoutables par modernizr

```
<html lang="en" dir="ltr" id="modernizr-com" class=" js flexbox canvas
canvastext webgl no-touch geolocation postmessage websqldatabase no-indexeddb
hashchange history draganddrop websockets rgba hsla multiplebgs backgroundsize
borderimage borderradius boxshadow textshadow opacity cssanimations csscolumns
cssgradients cssreflections csstransforms no-csstransforms3d csstransitions
fontface video audio localstorage sessionstorage webworkers applicationcache
svg inlinesvg smil svgclippaths">
```

Les classes CSS

À quoi servent toutes ces classes ajoutées par Modernizr 2 ? Tout simplement à fournir du design alternatif si les fonctionnalités ne sont pas supportées par le navigateur de l'utilisateur. Ainsi, pour pouvoir utiliser par exemple les images de fond multiples en CSS3 tout en proposant une solution alternative aux navigateurs ne les supportant pas, nous utiliserons le code suivant :

Utiliser les classes pour faire une alternative

```
.multiplebgs header {
    /* Les backgrounds multiples sont supportés */
}
.no-multiplebgs header {
    /* Les backgrounds multiples ne sont pas supportés */
}
```

Il est donc très simple de proposer une solution de secours pour les navigateurs les moins avancés dans le support des nouvelles fonctionnalités. Ces classes ne concernent pas uniquement les propriétés CSS3, elles permettent également d'appliquer un design différent en cas d'absence de géolocalisation, de balise `video`, des WebSockets... Bref, à moins que vous ne souhaitiez détecter le support d'une fonctionnalité très récente et expérimentale, Modernizr devrait détecter tout ce dont vous avez besoin.

Enfin, il est proposé de mettre une classe `no-js` sur votre élément `html`, que Modernizr remplacera par la classe `js`, ce qui permet également de proposer un design alternatif lorsque le JavaScript est désactivé.

Note : la version 2 de Modernizr permet de sélectionner en détail les fonctionnalités que l'on souhaite tester, ce qui allège considérablement les traitements effectués ainsi que la taille de la bibliothèque.

HTML5 Boilerplate

Si vous souhaitez démarrer un nouveau projet HTML5, vous allez commencer par établir la structure de votre page, ajouter toutes les bibliothèques dont vous aurez besoin (Modernizr, jQuery...), éventuellement préparer un fichier de configuration serveur comme le `.htaccess` en PHP, appliquer un CSS Reset pour annuler les styles par défaut des navigateurs... Bref, vous allez passer par une phase de préparation de l'architecture de votre site et vous devrez maximiser la compatibilité avec tous les navigateurs.



Logo de HTML5 Boilerplate.

Afin de nous faciliter cette pénible tâche, Paul Irish de Google (oui oui, notre préfacer !), ainsi que d'autres acteurs majeurs de la communauté HTML5 tels que Divya Manian, Nicolas Gallagher et Mathias Bynens se sont chargés de créer un gabarit de démarrage ultra-complet pour nous. En plus d'appliquer une multitude de bonnes pratiques pour la compatibilité HTML5 de votre site, Boilerplate inclut également des techniques particulièrement astucieuses telles que cibler certaines versions spécifiques d'Internet Explorer dans le CSS *via* des commentaires conditionnels.

Détection de la version de IE

```
<!--[if lt IE 7 ]> <html lang="fr" class="no-js lt-ie9 lt-ie8 lt-ie7"> <![endif]-->
<!--[if IE 7 ]> <html lang="fr" class="no-js lt-ie9 lt-ie8"> <![endif]-->
<!--[if IE 8 ]> <html lang="fr" class="no-js lt-ie9"> <![endif]--><!--[if gt IE 8]><!--> <html class="no-js" lang="en">
<!--<![endif]-->
```

Ainsi, à la manière de Modernizr, nous pouvons appliquer des styles à une version spécifique d'Internet Explorer voire même cibler « Toutes les versions antérieures à Internet Explorer 9, 8 ou 7 » grâce à `lt-ie9`, `lt-ie8` et `lt-ie7`, ce qui s'avère très pratique puisqu'en général les versions plus anciennes d'Internet Explorer ont besoin des mêmes patches que les récentes.

Appliquer des patches pour IE6

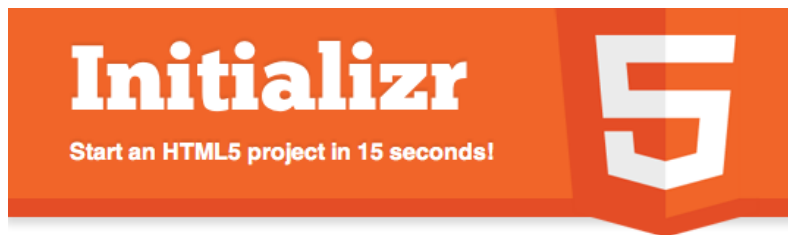
```
.lt-ie8 #logo{
  margin-left:5px;
}
```

HTML5 Boilerplate contient de nombreux outils très pratiques comme celui-ci afin que votre site soit prêt à être mis en production sur des bases solides.

Le seul ennui c'est qu'HTML5 Boilerplate est tellement complet qu'il en devient surchargé, et que chaque création de projet passe par une longue phase de suppression des fonctionnalités que vous n'utiliserez jamais.

Nous en venons donc à présenter Initializr.

Initializr



Logo d'Initializr

Initializr est un générateur de gabarit basé sur HTML5 Boilerplate qui vise à simplifier et accélérer le démarrage d'un projet HTML5.

Il a été créé par celui qui rédige actuellement ces lignes (Jonathan Verrecchia). Il s'agit donc pour une fois d'un projet français ! Vous n'aurez donc aucune difficulté à me contacter pour discuter du projet si l'envie vous en dit.

L'idée d'Initializr est de personnaliser le *template* de Boilerplate en y incluant uniquement ce dont vous avez besoin afin de démarrer très rapidement un nouveau projet.

Vous pouvez par exemple choisir d'utiliser Modernizr ou juste HTML5shiv, inclure jQuery en version compressée ou développement, si vous souhaitez Google Analytics sur votre page, ainsi qu'une multitude d'autres options.

HTML/CSS Template	HTML5 Polyfills	Server Config
☒ No template	☒ Modernizr	☒ .htaccess
☐ Mobile-first Responsive	☐ Just HTML5shiv	☐ nginx.conf
☐ Responsive Bootstrap 2.0.2	☐ Respond - Alternatives	☐ web.config

jQuery	Stylesheet Language	Google Services
☒ Minified	☒ CSS	☒ Chrome Frame
☐ Development	☐ LESS - Tutorial	☒ Analytics

H5BP Optional

☐ Build Script (6MB)	☒ Apple Touch Icons	☒ Humans.txt
☒ IE Classes	☒ plugins.js	☒ 404 Page
☒ Favicon	☒ Robots.txt	☒ Adobe Cross Domain

Exemples d'options de personnalisation d'Initializr.

Vous apprécierez également la possibilité de démarrer avec une structure et des styles par défaut (cela évite le syndrome de la page blanche !). Le gabarit proposé par Initializr est dit *responsive* (ou « adaptatif » en français), c'est-à-dire qu'il s'adapte à la taille de l'écran de l'utilisateur. Une même page s'affichera donc différemment sur un navigateur de bureau ou sur un navigateur mobile.



Le gabarit adaptatif d'Initializr sur un écran large

Pour les petits curieux parmi vous qui se demandent comment cela fonctionne, le gabarit utilise les *media queries*, une fonctionnalité CSS3 très pratique qui permet d'appliquer des styles différents selon la largeur de l'écran (ou de nombreux autres paramètres !). CSS3 sort du scope de ce livre donc nous vous invitons à faire quelques recherches en ligne, vous n'aurez aucun mal à trouver des informations sur le sujet si cela vous intéresse.



article header h1

Lorem ipsum dolor sit amet,
consectetur adipiscing elit. Aliquam

Le gabarit d'Initializr sur un mobile.

Avec tous ces nouveaux outils en main, vous avez de quoi démarrer votre prochain projet HTML5 sur des bases solides !

TABLES DE COMPATIBILITÉ

Il est parfois bien difficile de savoir ce qui appartient à la réalité ou ce qui reste de la fiction avec toutes ces nouvelles API HTML5. Comment savoir si l'History API fonctionnera sur un iPhone, ou quelle version de Firefox supporte WebGL ? Pour répondre à la question « mais où est ce que ce truc marche ? » voici deux tables de compatibilité :

- **En anglais**, caniuse.com par Alexis Deveria, très pratique, toujours à jour, et qui inclut même la compatibilité des navigateurs de smartphone.
- **En français** : html5demo.braincracking.org qui contient une table de compatibilité, et surtout des exemples qui marchent, exécutables dans le navigateur avec bien sûr mise à disposition du code source pour faire de vils copier / coller. Pour chaque exemple Frédéric Madsen a mis des notes d'implémentation, et des bugs qu'il a trouvés en faisant ses tests.

Pour continuer à se former après la lecture de ce livre :

- En français vous avez html5-css3.fr et braincracking.org qui vous proposent des tutoriaux ou des actualités. Leurs auteurs sont absolument géniaux, ce qui n'a rien à voir avec le fait qu'ils écrivent aussi ces lignes.
- Alsacreations.com pour ses forums et ses articles de fond, autant que pour ses auteurs, très actifs dans la formation.
- En anglais, html5doctor.com est un incontournable au niveau de la sémantique des nouveaux éléments, à tel point que leurs auteurs influencent les spécifications.
- html5rocks.com (Google), hacks.mozilla.org (Mozilla) et le blog de IE 10 (Microsoft) sont aussi d'excellentes ressources pour découvrir les nouveautés sous forme de news ou de tutoriaux, écrits par ceux-là même qui développent et implémentent les standards.

Le mot de la fin

Pour terminer nous aimerions vous remercier personnellement, car le simple fait que vous lisiez ces lignes fait que vous participez à l'engouement général pour HTML5. Une technologie émergente telle que celle-ci a besoin de soutien, d'experts, d'enthousiastes, et vous faites visiblement partie de l'aventure ! N'hésitez pas à promouvoir cette technologie autour de vous en répandant la bonne parole. Chaque nouveau converti participe à la sensibilisation générale, et par conséquent à l'adoption générale des navigateurs modernes.

En combinant les parts de marché d'Internet Explorer 9, Firefox, Chrome et Opera, au moins la moitié de vos utilisateurs peuvent déjà utiliser les API HTML5. Pour les anciens navigateurs, ce livre vous a donné des indications pour émuler les fonctionnalités.

Sur les mobiles, les leaders du marché des *smartphones* que sont Apple et Google proposent depuis longtemps un support tout à fait viable de nombreuses fonctionnalités particulièrement dédiées au monde mobile telles que la géolocalisation et l'offline. L'HTML5 ce n'est pas le futur, c'est bien le présent, et il est grand temps de s'y mettre !

Allez, puisqu'on est vraiment très très gentils, voici un dernier petit rappel pour la route :

```
<!doctype html>
```

ANNEXE

Fiches pratiques

Voici une série de fiches « résumé » des API ou des nouveautés HTML5 qui vous servira de référence en cas d'oubli du nom d'une propriété ou d'une méthode, en vous présentant des exemples de code typiques. Vous y trouverez également un lien direct vers chaque spécification, un tableau du support des navigateurs (daté d'avril 2011) et un rappel sur la méthode de contournement quand elle existe.

ARIA

- Spécification

URL	http://www.w3.org/TR/wai-aria/
Statut	Candidate Recommendation

- Support

Internet Explorer	Firefox	Safari	Chrome	Opera	Webkit iOS	Opera mobile	Webkit Android
8*	3.5	4*	8*	10.6*	3.2*	10*	X

* Support partiel. Voir <http://www.paciellogroup.com/blog/aria-tests/>
ARIA-SafariOperaIEFF-update2.html.

- Exemples

Déclarer la fonction d'une zone

```
<article role="main">
  Ceci est le contenu principal
</article>
```

Marquer une zone mise à jour dynamiquement

```
<div aria-live="polite" aria-atomic="true">
  Vous avez <span id="number">3</span> messages
</div>
```

Les trois valeurs de `aria-live` : `off` (par défaut), `polite` et `assertive`. L'attribut `aria-atomic` fait relire la `div` entière plutôt que seulement le `span` qui change.

- Rôles

Liste des rôles de zone et correspondance avec les balises HTML5.

Rôle	Unique ou multiple	Balise correspondante
<code>banner</code>	Unique	<code>header</code> , si relative au document
<code>contentinfo</code>	Unique	<code>footer</code> , si relative au document
<code>complementary</code>	Multiple	<code>aside</code> , si cette balise est au même niveau hiérarchique que le nœud avec le <code>role="main"</code>
<code>navigation</code>	Multiple	<code>nav</code>
<code>form</code>	Multiple	<code>form</code>
<code>search</code>	Multiple	<code>form</code>
<code>main</code>	Unique	-
<code>application</code>	Multiple	<code>body</code> , <code>svg</code> , <code>embed</code> , <code>video</code> ...
<code>document</code>	Multiple	<code>body</code> par défaut

Liste des rôles de widget ARIA.

Rôles	Fonctionnalité
<code>dialog</code> , <code>alertdialog</code>	boîtes de dialogue
<code>log</code> , <code>marquee</code> , <code>status</code> , <code>timer</code>	zones de texte mises à jour en continu
<code>menuitem</code> , <code>menu</code> , <code>menubar</code> , <code>menuitemradio</code> , <code>menuitemcheckbox</code>	menus
<code>scrollbar</code> , <code>slider</code>	barres de défilement non natives, curseurs
<code>progressbar</code>	indicateurs de progression
<code>tabpanel</code> , <code>tablist</code> , <code>tab</code>	onglets
<code>tooltip</code>	tooltips
<code>treeitem</code>	arbres hiérarchisés comme dans un explorateur de fichiers

- Debug

Voir les zones : <https://addons.mozilla.org/en-US/firefox/addon/juicy-studio-accessibility-too/>

CANVAS 2D

- Spécification

URL	http://www.w3.org/TR/2dcontext/
Statut	Draft

- Support

Internet Explorer	Firefox	Safari	Chrome	Opera	Webkit iOS	Opera mobile	Webkit Android
9.0	3.0	3.0	3.0	10.0	1.0	10.0	1.0

- Exemples

Déclaration du canvas

```
<canvas id="mycanvas" width="500" height="300">
```

Le code HTML présent ici s'affiche si le navigateur ne supporte pas canvas

```
</canvas>
```

Structure de base habituelle pour dessiner dans un canvas

```
var oCtx = null;
window.addEventListener('load', function() {
  oCtx = document.getElementById("mycanvas").getContext("2d");
  dessinerUnTruc();
}, false);
function dessinerUnTruc(){
  // Votre fonction de dessin
};
```

Tracer un rectangle rouge

```
function dessinerRectangleRouge(){
  oCtx.fillStyle = "red";
  // trace un rectangle au point (100, 100) de largeur 300 et hauteur 50
  oCtx.fillRect(100, 100, 300, 50);
};
```

Tracer un triangle à bords épais noirs et contenu gris

```
function dessinerTriangleFerre(){
    // début du tracé
    oCtx.beginPath();
    oCtx.moveTo(100, 100);
    // tracé des 3 côtés
    oCtx.lineTo(200, 200);
    oCtx.lineTo(200, 100);
    oCtx.lineTo(100, 100);
    // fermeture du tracé
    oCtx.closePath();
    // styles
    // épaisseur de la ligne de contour
    oCtx.lineWidth = 10;
    // couleur du contenu
    oCtx.fillStyle = "#ddddd";
    // affichage du contenu
    oCtx.fill();
    // couleur du contour
    oCtx.strokeStyle = "#222222";
    // affichage du contour
    oCtx.stroke();
}
```

Effectuer une translation de contexte

```
function dessinerAvecTranslation(){
    // sauvegarde de la position initiale du contexte
    oCtx.save();
    // translation du contexte
    oCtx.translate(250, 150);
    // effectuer le tracé
    // restauration de la position initiale du contexte
    oCtx.restore();
}
```

Ombrer un dessin

```
// le décalage horizontal
oCtx.shadowOffsetX = 5;
// le décalage vertical
oCtx.shadowOffsetY = 5;
// le rayon de flou
oCtx.shadowBlur = 10;
// la couleur
oCtx.shadowColor = 'rgba(0, 0, 0, 0.3)';
```

Utiliser la Text API

```
// on applique la propriété CSS souhaitée
oCtx.font = "30px Calibri";
// on centre le texte
oCtx.textAlign = "center";
// on l'affiche aux coordonnées (0, 0) avec une largeur maximale de 200
oCtx.fillText("Coucou", 0, 0, 200);
```

Styliser avec un dégradé

```
// initialisation du dégradé allant de (50, 50) à (100, 100)
var oGradient = oCtx.createLinearGradient(50, 50, 100, 100);
// déclaration des points d'arrêt des couleurs
oGradient.addColorStop(0, "white");
oGradient.addColorStop(1, "black");
// attribution du dégradé au style sur l'élément souhaité
oCtx.fillStyle = oGradient;
```

Charger et afficher une image

```
function dessinerLogo(){
    var oLogo = new Image();
    oLogo.src = "logohtml5.png";
    // fonction lancée uniquement une fois l'image chargée
    oLogo.onload = function(){
        oCtx.drawImage(oLogo, 120, 20);
    };
}
```

Utiliser cette image en fond répété

```
oCtx.fillStyle = oCtx.createPattern(oLogo, "repeat");
```

Récupérer le tableau de pixels de toute la surface du canvas

```
var oImageData = oCtx.getImageData(0, 0,
                                     oCtx.canvas.width,
                                     oCtx.canvas.height);
```

La structure d'un tableau de pixels est la suivante :

[r1, g1, b1, a1, r2, g2, b2, a2, r3, g3, b3, a3, ...]

Toutes les valeurs vont de 0 à 255.

Exportation du contenu du canvas en PNG lors du clic sur « save »

```
document.getElementById("save").addEventListener('click', function() {
    window.location = oCtx.canvas.toDataURL("image/png");
}, false);
```

Réaliser une animation d'un bloc qui se déplace vers la droite

```
// initialisation des variables
var oCtx = null;
var oRect = {x:0, y:100};
window.addEventListener('load', function() {
  oCtx = document.getElementById("mycanvas").getContext("2d");
  setInterval(mainLoop, 16);
}, false);
// boucle principale déclenchée toutes les 16 millisecondes
function mainLoop(){
  // on efface l'affichage précédent
  oCtx.clearRect(0, 0, 500, 300);
  // on place le nouveau rectangle (qui est de largeur 100 et hauteur 60)
  oCtx.fillRect(oRect.x, oRect.y, 100, 60);
  // on le fait avancer d'un pixel vers la droite
  oRect.x++;
}
```

- **Émulation**

Il est possible de faire fonctionner [canvas](#) sur Internet Explorer 7 et 8 grâce à ExplorerCanvas, basé sur VML, ou FlashCanvas qui utilise Flash.

FILE API

- **Spécification**

URL	http://www.w3.org/TR/FileAPI/
Statut	Working Draft

- **Support**

Internet Explorer	Firefox	Safari	Chrome	Opera	Webkit iOS	Opera mobile	Webkit Android
10.0	3.6	6	8	11.1	X	11.1	3.0

- **Exemples**

Tester le support navigateur

```
var oEl = document.createElement('input');
oEl.type = 'file';
return ('files' in oEl); // true ou false
```

Accéder aux fichiers sélectionnés ou jetés

```
<input id="user_file" type="file" multiple />
<div id="drop-zone"> Jeter ici </div>
<script>
```

```

document.getElementById('user_file').onchange = function() [
    afficherFichiers( this.files );
];
document.getElementById('user_file').ondrop = function( e ) [
    afficherFichiers( e.dataTransfer.files );
];
// fonction générique de récupération d'un objet FileList
function afficherFichiers( oFiles ) {
    for(var i = 0; i < oFiles.length; i++) {
        console.log(oFiles[i].name, oFiles[i].size);
    }
};
</script>

```

- **Émulation**
Pas d'émulation possible.

FORMULAIRES 2.0

- **Spécification**

URL	http://www.w3.org/TR/html5/forms.html
Statut	Working Draft

- **Support**

Internet Explorer	Firefox	Safari	Chrome	Opera	Webkit iOS	Opera mobile	Webkit Android
10.0	4.0*	5.0*	6.0*	10.6*	4.0*	10.0*	2.3

* Aucun navigateur n'a un support complet. Voir <http://wufoo.com/html5/>

- **Exemples**

Un formulaire utilisant les nouveaux types

```

<input type="search" />
<input type="tel" />
<input type="email" />
<input type="url" />
<input type="date" />
<input type="number" />
<input type="range" />
<input type="color" />

```

Votre nouvelle trousse à outil pour créer des formulaires. `number` correspond au « champ avec des petits boutons à droite pour augmenter et réduire », et `range` au *slider*.

Champ numérique allant de 0 à 100 par pas de 5 en partant de 50

```
<input type="number"
      min="0"
      max="100"
      step="5"
      value="50">
```

Les attributs `min`, `max` et `step` permettent d'indiquer les valeurs minimales et maximales ainsi que le pas aux champs de type `number`, `range` et `date`. `value` est la valeur actuelle, ainsi que celle de départ.

Créer un datalist

```
<input type="text" list="serie">
<datalist id="serie">
  <option value="Les Simpsons">
  <option value="South Park">
  <option value="Heroes">
</datalist>
```

Crée un champ proposant des suggestions à l'utilisateur à la manière des suggestions d'un moteur de recherche.

Personnaliser le look des champs en CSS

```
:required{ /* champ requis */ }
:optional{ /* champ optionnel */ }
:valid{ /* valeur valide */ }
:invalid{ /* valeur invalide */ }
:in-range{ /* valeur contenue entre min et max */ }
:out-of-range{ /* valeur en dehors de min et max */ }
:default{ /* lorsque la valeur n'a pas été modifiée */ }
```

Désactiver la validation native d'un formulaire

```
<form novalidate>
```

Rendre un champ obligatoire

```
<input name="mail" type="email" required>
```

Appliquer une regex de validation de donnée

```
<input type="text" pattern="[A-Z][a-z]+-[A-Z][a-z]+" />
```

Sélectionner automatiquement un champ à l'ouverture de la page

```
<input type="search" autofocus>
```

Utiliser un texte indice dans un champ

```
<input type="search" placeholder="Entrez votre recherche">
```

Le *placeholder* apparaît uniquement lorsque le champ est vide et non sélectionné.

Sélection multiple de fichier et filtre sur le type mime

```
<input type="file" multiple accept="image/*, video/*">
```

- **Émulation**

Les bibliothèques H5F et Webforms2 vous permettent d'émuler le comportement des formulaires HTML5 en JavaScript.

GÉOLOCALISATION

- **Spécification**

URL	http://www.w3.org/TR/geolocation-API/
Statut	Candidate Recommendation

- **Support**

Internet Explorer	Firefox	Safari	Chrome	Opera	Webkit iOS	Opera mobile	Webkit Android
9.0	3.5	5.0	5.0	10.6	3.0	11.0	2.0

- **Exemples**

Récupération minimaliste de la position d'un utilisateur

```
navigator.geolocation.getCurrentPosition(function(position){  
    alert(position.coords.latitude + " " + position.coords.longitude);  
});
```

Ce code est la plus courte manière de « voir si ça marche ».

Récupération de la position avec gestion des erreurs

```
navigator.geolocation.getCurrentPosition(successCallback, errorCallback);  
function successCallback(position){  
    alert(position.coords.latitude + " " + position.coords.longitude);  
};  
function errorCallback(error){  
    alert("Code d'erreur : " + error.code);  
}
```

Une version un peu plus modulaire et maintenable avec un début de gestion d'erreurs.

Exemple d'utilisation du paramètre d'options

```
navigator.geolocation.getCurrentPosition(  
  successCallback,  
  errorCallback,  
  { enableHighAccuracy : true,  
    timeout : 10000,  
    maximumAge : 600000  
});
```

- `enableHighAccuracy` gère l'activation du GPS,
- `timeout` est le délai maximal de réponse du service et vaut par défaut `Infinity`,
- `maximumAge` est la durée de validité d'une position et vaut par défaut `0`.
- `timeout` et `maximumAge` sont en millisecondes.

Déclencher et arrêter le suivi de la position de l'utilisateur

```
var iWatchId = navigator.geolocation.watchPosition(successCallback,  
errorCallback);  
navigator.geolocation.clearWatch(iWatchId);
```

Tant que `clearWatch()` n'est pas appelé, le `successCallback` est déclenché à chaque fois que l'utilisateur change de position. `watchPosition()` peut également prendre un troisième paramètre d'options, exactement comme dans l'exemple précédent.

Attributs de géolocalisation obtenus

```
function successCallback(position){  
  alert(position.coords.latitude);  
  alert(position.coords.longitude);  
  alert(position.coords.altitude); (mètres)  
  alert(position.coords.accuracy); // la précision (mètres)  
  alert(position.coords.altitudeAccuracy); // la précision d'altitude (mètres)  
  alert(position.coords.heading); // l'angle par rapport au nord (de 0 à 360)  
  alert(position.coords.speed); // la vitesse (mètres par seconde)  
  alert(position.timestamp); // l'heure à laquelle a été obtenue la position  
};
```

- **Émulation**

Il est toujours possible d'utiliser la géolocalisation par IP si vous n'avez pas besoin d'une position de très haute précision de type GPS.

MICRODATA

- Spécification

URL	http://www.w3.org/TR/microdata/
Statut	Draft

- Support

Internet Explorer	Firefox	Safari	Chrome	Opera	Webkit iOS	Opera mobile	Webkit Android
X	5	X	12	12	X	X	X

* Microdata servant surtout aux programmes externes, le support navigateur est anecdotique.

- Exemples

Trois manières de marquer les valeurs des propriétés d'un objet

```
<div itemscope>
  Ceci est un objet qui a plusieurs propriétés
  dont name qui a pour valeur <span itemprop="name">Mon nom</span>
  image qui a pour valeur le href de cette image :
  
  et enfin date qui a pour valeur l'attribut datetime de l'élément time :
  <time itemprop="date" datetime="2011-04-15">le 15 avril</time>
</div>
```

Se rattacher au vocabulaire hCard de microformat

```
<div id="presentation"
  itemscope
  itemtype="http://microformats.org/profile/hcard">
  >
   Salut, j'aime la vie et je m'appelle
  <span itemprop="given-name"> Kévina </span>.
  J'ai né <time itemprop="bday" datetime="1996-04-08">il y a 15 ans</time> et
  j'adore mon chat KikouLo! .
  Envoyez nous <a itemprop="email" href="mailto:kevinaDu62@skaïblog.com">un
  message</a> !
</div>
```

Chercher les objets hCard de sa propre page

```
// on va chercher tous les objets hCard de la page
var oList = document.getItems('http://microformats.org/profile/hcard'),
    aProperties;
for(var i=0 ; i < oList.length ; i++) {
```

```

aProperties = oList[i].properties ;
// on obtient une liste des propriétés de l'objet
for(var j = 0 ; j < aProperties.length ; j++) {
  if(aProperties[j].itemProp == 'given-name') {
    console.log(aProperties[j].itemValue ) ; // on a retrouvé Kévina !
  }
}
}
}

```

Associer à l'objet des informations qui ne sont pas ses enfants

```

<header>
  <a href="http://braincracking.org" id="home" itemprop="url">Accueil</a>
</header>
...
<div itemscope itemtype="http://microformats.org/profile/hcard"
  itemref="email-id home">
  <span itemprop="fn">Jean-pierre VINCENT</span>
</div>
...
<footer>
  Contact: <span id="email-id" itemprop="email">jp@braincracking.org</span>
</footer>

```

On utilise `itemref` qui permet de lister les `id` à rattacher à l'objet.

- **Émulation**

L'API est simulable en JavaScript, avec une librairie du nom de `microdata.js` :
<http://gitorious.net/microdatajs/microdatajs/blobs/master/microdata.js>

- **Debug**

Tester le *markup* :

- <http://foolip.org/microdatajs/live/>
- <http://www.google.com/webmasters/tools/richsnippets>

Générer un *markup* : <http://microdata.freebaseapps.com/index>

- **Vocabulaires**

Voir les fiches qui commencent par un `h` : <http://microformats.org/profile/>

Le site du vocabulaire Microdata de Google : <http://www.data-vocabulary.org/>

MULTIMÉDIA

- **Spécification**

URL	http://www.w3.org/TR/html5/video.html
Statut	Working Draft

- Support

	Internet Explorer	Firefox	Safari	Chrome	Opera	Webkit iOS	Opera mobile	Webkit Android
Balises	9	3.5	4	8	10.6	4.1	11	2.1
H.264	9	X**	3.2	8	X	3.2	X	2.1
VP8 / Theora	9*	4	X	8	10.6	X	11	2.3
MP3	9	X	9	8	X	3.2	X	?
Vorbis	X	3.6	X	8	10.6	X	11	?

* IE ne lira VP8 que si l'utilisateur a installé lui-même les codecs.

** Firefox mobile lira nativement le H.264

- Exemples

Vidéo avec toutes les options

```
<video src="video.url" controls loop autoplay preload="metadata" width="854"
height="480" poster="image.png">
  Votre navigateur est démodé
</video>
```

Audio avec toutes les options

```
<audio src="audio.url" controls loop autoplay preload="metadata">
  Votre navigateur est démodé
</audio>
```

L'intérieur des balises multimédia n'est affiché que si le navigateur ne connaît pas les balises. Pour la détection du format, il faut le faire en JavaScript.

L'attribut `preload` peut prendre les valeurs `auto`, `none` et `metadata`.

Détection du support des balises

```
(!!document.createElement('video').canPlayType); // true ou false
```

Détection du support d'un format

```
var oElement = document.createElement('video');
if( oElement.canPlayType(
  'video/mp4; codecs="avc1.42E01E, mp4a.40.2"'
) === 'probably' ){
  return true;
}
```

Il vaut mieux préciser le codec en plus du type mime, sous peine d'avoir la réponse `'maybe'`, qui est trop ambiguë. Une réponse `''` (chaîne vide) est un non.

```
Distribuer une vidéo selon la taille d'écran
<video controls>
  <source src="video.hires.url"
    media="(min-device-width :800px)"></source>
  <source src="video.lowres.url"></source>
</video>
```

Laisser le navigateur choisir son format de vidéo

```
<video controls>
  <source src="video.mp4" type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"'>
  <source src="video.webm" type='video/webm; codecs="vp8, vorbis"'>
</video>
```

- Événements importants

Réception d'informations sur le média

```
oVideo.onloadedmetadata = function( ) {
  oVideo.duration ; // durée totale en secondes
  oVideo.videoWidth ; // réelle largeur de la vidéo
};
```

La vidéo peut être lue sans interruption

```
oVideo.oncanplaythrough = function() {
  oVideo.play();
};
```

Calcul de la position de la tête de lecture

```
oVideo.ontimeupdate = function() {
  console.log(oVideo.currentTime / oVideo.duration );
};
```

Calcul du pourcentage du fichier téléchargé

```
oVideo.onprogress = function() {
  var iBuffered = oVideo.buffered.end( 0 );
  console.log( iBuffered / oVideo.duration );
};
```

- **Émulation**

Avec Flash sur les navigateurs qui ne supportent pas les balises `audio` / `video` ou qui ne supportent pas un autre format que le H.264 et le MP3.

Comparatif de la vingtaine de librairies existantes :

<http://praegnanz.de/html5video/>

L'OFFLINE (APPCACHE)

- **Spécification**

URL	http://www.whatwg.org/specs/web-apps/current-work/multipage/offline.html
Statut	WHATWG

- **Support**

Internet Explorer	Firefox	Safari	Chrome	Opera	Webkit iOS	Opera mobile	Webkit Android
10.0	3.5	4.0	5.0	10.6	2.1	11.0	2.1

- **Exemples**

Spécifier l'utilisation d'un fichier manifeste

```
<html manifest="appcache.appcache">
```

Déclarer le MIME-type des fichiers manifest avec Apache

```
AddType text/cache-manifest appcache
```

Fichier manifeste typique

```
CACHE MANIFEST
# v0.1
CACHE:
cached.html
css/style.css
img/superlogo.png
FALLBACK:
/ offline.html
NETWORK:
*
```

- La section **CACHE** : liste les fichiers à mettre en cache,
- La section **FALLBACK** : liste les fichiers qui doivent se rabattre sur d'autres si la connexion internet est coupée,
- La section **NETWORK** : liste les fichiers nécessitant une connexion internet, ce qui est vital pour pouvoir accéder aux ressources externes.

Les événements de la Cache API JavaScript sont :

- **onchecking** : est déclenché quand le navigateur commence à vérifier si des mises à jour sont disponibles.
- **onerror** : est déclenché lorsqu'une erreur se produit (si l'un des fichiers spécifié dans le manifeste est manquant par exemple).
- **onnoupdate** : est déclenché s'il n'y a pas de nouvelle version disponible.
- **ondownloading** : est déclenché au début du téléchargement de l'ensemble des fichiers.
- **onprogress** : est déclenché au début du téléchargement de chaque fichier.
- **onupdateready** : est déclenché à la fin du téléchargement de l'ensemble des fichiers.
- **Oncached** : est déclenché lorsque le téléchargement est terminé et que le cache est prêt à être utilisé.
- **Onobsolete** : est déclenché lorsque le manifeste est introuvable.

Détecter un changement de cache et le mettre à jour

```
setInterval( window.applicationCache.update, 60000); // 1 minute
window.applicationCache.addEventListener("updateready", function(){
    window.applicationCache.swapCache();
}, false);
```

Note : une actualisation de la page sera toujours nécessaire afin de prendre en compte le nouveau cache.

SERVER SENT EVENT

- Spécification

URL	http://www.w3.org/TR/eventsourcing/
Statut	Working Draft

- Support

Internet Explorer	Firefox	Safari	Chrome	Opera	Webkit iOS	Opera mobile	Webkit Android
X	9	5	8	11	4.0	11.1	X

- Exemples

Écouter le serveur

```
var oConnexion = new EventSource('http://example.com/getdata.php');
oConnexion.onmessage = function( e ) {
    console.log( e.data );
};
```

Événements importants et utilisation des statuts

```
var oConnexion = new EventSource('http://example.com/getdata.php');
oConnexion.onerror = function( e ) {
    console.log('erreur'); // true
};
oConnexion.onopen = function( e ) {
    console.log('this.readyState === EventSource.OPEN'); // true
};
```

La propriété `readyState` peut aussi prendre la valeur de la constante `EventSource.CLOSED` ou `EventSource.CONNECTING`.

Événement nommé : chaîne envoyée par le serveur

```
event: tic-tac
data: Il est 23:45:54
id: 123456789
```

Événement nommé : objet interprété par le client

```
var oSource = new EventSource('http://example.com/getdata.php');
oSource.addEventListener('tic-tac', function( e ) {
    e.type; // 'tic-tac'
    e.lastEventId; // '123456789'
    e.data; // 'Il est 23:45:54'
}, false);
```

Le serveur peut préfixer ses lignes avec quatre valeurs :

- **data** : qui permet d'envoyer des données sur une seule ligne,
- **event** : qui permet de nommer des événements. JavaScript peut alors se brancher dessus comme un événement normal du DOM,
- **id** : qui peut être utilisé comme numéro de ticket en cas de déconnexion, car le navigateur renvoie à la prochaine connexion l'entête **Last-Event-ID** avec la dernière valeur reçue du serveur,
- **retry** : qui permet au serveur d'indiquer un délai avant reconnexion au client.

- Côté serveur

Détecter le support navigateur en PHP

```
$hasSSE = ($_SERVER['HTTP_ACCEPT'] === 'text/event-stream'); // true ou false
```

Envoyer une information au client en PHP

```
header('Content-Type: text/event-stream');
// boucle infinie pour garder la connexion ouverte
while( true ) {
    // méthode métier de récupération de nouvelles données (base, fichier
    plat...)
    $data = json_encode( get_data() );
    print 'data: '. $data ;
    print PHP_EOL.PHP_EOL; // double retour chariot indispensable
    sleep(1);
}
```

Dans la même boucle infinie, on peut supporter une technique de *long polling* avec *iframe* cachée en envoyant les mêmes données dans des balises **script**.

- Émulation

Server Sent Event remplace les techniques de *long polling* à base d'*iframe* cachée, on peut donc continuer à utiliser cette technique lorsque le navigateur ne supporte pas l'API.

Il existe une librairie YUI3 : <http://yuilibrary.com/gallery/show/eventsources>

WEB STORAGE

- Spécification

URL	http://www.w3.org/TR/webstorage/
Statut	Candidate Recommendation

- Support

Internet Explorer	Firefox	Safari	Chrome	Opera	Webkit iOS	Opera mobile	Webkit Android
8	3.5	4	8	10.6	3.2	11	2.1

- Exemples

Permanent ou pas ?

```
window.sessionStorage.setItem('test', 'temporaire');  
window.localStorage.setItem('test', 'permanent');
```

À la fermeture de la fenêtre, le contenu de `sessionStorage` est vidé.

Définir, retrouver, supprimer une clé

```
localStorage.setItem('locale', 'fr-FR');  
console.log( localStorage.getItem('locale') ); // fr-FR  
localStorage.removeItem('locale');
```

Tout supprimer

```
localStorage.clear();
```

La valeur des clés est accessible depuis toutes les pages du domaine.

Chaîne vide ou valeur jamais définie ?

```
if( localStorage.getItem('locale') === null )  
    console.log('locale n'a jamais été définie');
```

L'espace de stockage est-il vierge ?

```
if( localStorage.length === 0 )  
    console.log('Like a virgin');
```

Utiliser JSON pour retrouver les types et sauvegarder des structures

```
Var oLocale = {  
    Language : 'fr',  
    Country : 'FR'  
};  
localStorage['locale'] = JSON.stringify( oLocale );  
(typeof localStorage['locale'] ) // String  
var oStoredLocale = JSON.parse( localStorage['locale'] );  
(oStoredLocale.country); // FR
```

Sans cela, Web Storage ne sait conserver que des `String`.

Détecter une modification de valeur

```
window.addEventListener('storage', function( e ) {
  console.log( e.key + ' a une nouvelle valeur : ', e.newvalue);
, false);
```

Peut-être détourné pour détecter que le site est ouvert dans une autre fenêtre et les faire communiquer entre elles.

Détecter un dépassement de limite

```
try {
  localStorage['language'] = 'fr-FR';
} catch (e) {
  if (e == QUOTA_EXCEEDED_ERR)
    console.log("c'est trop énorme");
}
```

- **Émulation**

La fonctionnalité existe depuis longtemps sous différentes formes :

- IE6-7 : *userData*, avec 64 Ko de données,
- FF 2 et 3 : *globalStorage*,
- IE8 : Web Storage avec 10 Mo de données, mais avec une API asynchrone,
- Flash Shared Object, pour tout le monde avec 100 Ko de données,
- `window.name` comme hack universel pour émuler `sessionStorage`.

Trois bibliothèques d'abstraction d'accès à toutes ces méthodes existent :

- `jStorage` (www.jstorage.info), pour `jQuery`,
- `YUI2 Storage`, pour `YUI2`,
- `YUI3 Storage Lite`, si vous n'utilisez aucune de ces deux bibliothèques.

SYNTAXE ET NOUVELLES BALISES

- **Spécification**

URL	http://www.w3.org/TR/html5-diff/
Statut	Working Draft

- **Support**

Internet Explorer	Firefox	Safari	Chrome	Opera	Webkit iOS	Opera mobile	Webkit Android
9.0	3.5	3.2	5.0	10.0	3.2	10.0	2.1

- Exemples

Code de démarrage typique d'un projet HTML5

```
<!doctype html>
<html lang="fr">
<head>
  <title></title>
  <meta charset="UTF-8" />
  <link href="style.css" rel="stylesheet" />
  <script src="script.js"></script>
</head>
<body>
</body>
</html>
```

Syntaxes équivalentes

```
<!-- Guillemets optionnels -->
<meta charset=utf-8>
<meta charset="utf-8">
<!-- / de fermeture optionnelle -->


<!---Insensibilité à la casse -->
<link rel="stylesheet" href="test.css">
<LiNk rEl="STYLesheet" hRef="TeST.cSS">
```

Spécifier des attributs booléens sans valeur

```
<video autoplay loop controls>
```

Utilisation de <time> avec une date complète

```
<time datetime="2011-04-01T18:26:31+01:00">18h26, Paris, 1er avril 2011</time>
```

Utilisation de <figure> et <figcaption>

```
<figure>
  
  
  
  <figcaption>Figure 2.3 - Les logos de la sémantique HTML5, des applications
offline, et des effets 3D</figcaption>
</figure>
```

Les nouvelles balises de structure sont `header`, `footer`, `nav`, `aside`, `article` et `section`, parmi lesquelles `article`, `section`, `aside` et `nav` sont « sectionnantes ».

- Émulation

Le support d'IE6, 7 et 8 peut facilement être obtenu grâce à l'utilisation d'HTML5shiv (voir chapitre de conclusion).

WEBSOCKETS

- Spécification

URL	http://www.w3.org/TR/websockets/
Statut	Candidate Recommendation

- Support

Internet Explorer	Firefox	Safari	Chrome	Opera	Webkit iOS	Opera mobile	Webkit Android
10.0	9	5	8	11.6	4.2	11	X

* Il y a eu une faille de sécurité pendant quelques mois qui a complètement disparu.

- Exemples

Écouter le serveur

```
var oConnection = new WebSocket('ws://example.com');
oConnection.onmessage = function(e) {
    console.log(e.data); // chaîne envoyée par le serveur
};
```

Envoyer un message

```
oConnection.send('PONG !');
```

Événements importants et utilisation des statuts

```
var oConnection = new WebSocket('ws://example.com');
console.log( oConnection.readyState === WebSocket.CONNECTING ); // true
oConnection.onopen = function(e) {
    console.log( this.readyState === WebSocket.OPEN ); // true
};
oConnection.onclose = function(e) {
    console.log( this.readyState === WebSocket.CLOSED ); // true
};
oConnection.onerror = function(e) {
    console.log( e ); // affiche le détail de l'erreur
};
```

La propriété `readyState` peut aussi prendre la valeur de la constante `WebSocket.CLOSING`.

- **Côté serveur**

Un serveur HTTP normal ne suffit pas, il faut un serveur gérant le protocole WebSockets. Il en existe pour plusieurs langages :

- PHP : `phpWebSockets` (<http://code.google.com/p/phpwebsockets/>).
- Java : `jWebSocket` (<http://jwebsocket.org/>).
- JavaScript:Socket.IO-node (<http://github.com/LearnBoost/Socket.IO-node>)

- **Émulation**

Il existe des bibliothèques d'abstraction qui utilisent WebSockets natif ou Flash :

- la bibliothèque fournie avec votre serveur WebSocket si elle existe
- `socket.IO Client` (<https://github.com/LearnBoost/Socket.IO-node-client>)
- `web-socket-js` (<https://github.com/gimite/web-socket-js>)

XMLHttpRequest LEVEL 2

- **Spécification**

URL	http://www.w3.org/TR/XMLHttpRequest2/
Statut	Draft

- **Support**

Internet Explorer	Firefox	Safari	Chrome	Opera	Webkit iOS	Opera mobile	Webkit Android
10.0	4.0	5.0	7	12	5	12	3

- **Exemples**

Aller chercher un contenu en ligne

```
var oXHR = new XMLHttpRequest();
oXHR.open('GET', sUrl);
oXHR.onload = function(e) {
    console.log(e.currentTarget.responseText);
};
oXHR.send( );
```

Ce code simple est compatible XHR1 (IE7 et 8).

Progression du téléchargement et de l'envoi

```
var oXHR = new XMLHttpRequest();
oXHR.open('GET', sUrl);
oXHR.onprogress = function(e) {
```

```

    if(e.lengthComputable)
        console.log( (e.loaded / e.total) * 100 );
};
oXHR.upload.onprogress = function(e) {
    console.log( (e.loaded / e.total) * 100 );
};
oXHR.send( );

```

Note : pour le téléchargement, le header `Content-Length` doit être envoyé par le serveur.

Envoyer un message texte au serveur

```

var oXHR = new XMLHttpRequest();
oXHR.open('HEAD', sUrl);
oXHR.setRequestHeader("Content-Type", "text/plain;charset=UTF-8");
oXHR.send( 'Texte libre' );

```

Utile pour *loguer* des erreurs au front, ou faire du *tracking* d'utilisation de l'interface.

Joindre un fichier

```

<input type="file" id="file_input" multiple >
<script>
var oXHR = new XMLHttpRequest();
XHR2Uploader.oXHR.open('POST', sUrl);
var oForm = new FormData( ),
    oFileInput = document.getElementById('file_input');
oFormData.append( oFileInput.name, oFileInput.files[0] );
XHR2Uploader.oXHR.send( oFormData );
</script>

```

Le deuxième argument de `FormData.append()` est généralement une `String` mais on peut y mettre un pointeur vers un objet `File`.

- **Émulation**

Pour envoyer des fichiers et surveiller la progression : bibliothèques Flash ou applets java.

Pour recréer des formulaires dynamiquement, JavaScript.

Index

A

[accept](#) 131
ARIA 61
[audio](#) 68, 77
autofocus 46

C

Canvas 96
chat 185
[contentEditable](#) 74
CSS3 95

D

[datalist](#) 74
[details](#) 74
Doctype 13
drag and drop 132, 142

F

[figcaption](#) 71
[figure](#) 71
File 134
Flash Shared Object 180
forever iframe 194, 197
[FormData](#) 136

G

globalStorage 180

H

H.264 86
hiérarchie 69
[Hx](#) 69

I

IndexedDB 182

L

landmark roles 61
liens d'évitement 61
[localStorage](#) 175
long polling 197

M

manifeste 167
Microdata 51
Microformat 52
MP3 89
[multiple](#) 131

O

offline 165

Ogg 88

P

placeholder 46

polling 196

progress 72, 137

Proxys 191

R

RDFa 52

role 61

S

Server Sent Event 193

sessionStorage 175

SVG 94

T

Theora 87

types de champs 35

U

userData 180

V

validation 42

video 68, 77

W

WAI-ARIA 61

Web SQL Database 181

Web Storage 175

WebGL 98

WebM 87

WebSocket 183

X

XHR2 136

XHTML5 16

XMLHttpRequest Level 2 136