

Los API JavaScript de HTML5

Integre la potencia de HTML5
en sus aplicaciones Web

Luc VAN LANCKER



INFORMÁTICA TÉCNICA



Los API JavaScript de HTML5

Integre la potencia de HTML5 en sus aplicaciones Web

Este libro está dirigido a los **desarrolladores** de páginas Web que quieran aprovechar al máximo los **API JavaScript de HTML5**. El autor explora muchos de estos API JavaScript, algunos de ellos totalmente operativos, mientras que hay otros que todavía están en fase de desarrollo. HTML5 representa una evolución muy importante que cambia completamente el diseño de las páginas o aplicaciones Web. El autor ha tratado de adoptar un **enfoque pragmático y explicativo**, apoyándose en**numerosos ejemplos y capturas de pantalla**.

El objetivo del libro es doble: en primer lugar permite al lector integrar alguno de estos API en sus aplicaciones, como la geolocalización, el diseño 2D, el almacenamiento de datos en local o, por qué no, una base de datos. En segundo lugar permite descubrir el gran impulso que van a crear estos API JavaScript que serán, considerados globalmente, una verdadera plataforma de desarrollo de aplicaciones HTML5.

Los diferentes capítulos del libro detallan, de manera particular: el **API Selectors** que aporta soluciones a las carencias del JavaScript tradicional en la selección de los elementos del DOM – el API más mediático por el momento, **el API de geolocalización**, que permite conocer las coordenadas geográficas del usuario - **el API Storage** que permite conservar en el navegador los datos que se podrán utilizar más adelante sin tener que recurrir al servidor - **el API Offline** construido para permitir que los tablets y Smartphones puedan seguir utilizando **una aplicación en modo offline** como consecuencia, por ejemplo, de una pérdida de red - **el API History** que permite crear nuevas entradas en el histórico –**el API Drag & Drop** que permite usar la funcionalidad de **drag&drop** de manera nativa ... A todo esto sigue una serie de API más limitados como **la selección de archivos**, la posibilidad de **transmitir información** entre diferentes ventanas o etiquetas iframe, dentro del mismo dominio o de dominios diferentes, la ejecución de **scripts en segundo plano o de modo desatendido** y **el API WebSocket** que permite abrir una conexión bidireccional permanente entre el cliente y el servidor. Para terminar, **el API Canvas** que permite el diseño 2D directamente en la página sin tener que utilizar imágenes.

Los elementos adicionales se pueden descargar en este página.

Los capítulos del libro:

Introducción – Presentación – El API Selectors – La geolocalización – El almacenamiento de datos en local – El API Web SQL Database – El API Indexed Database – La edición del contenido (contentEditable) – El modo desconectado (offline) – Manipular el histórico del navegador – Drag & drop – La selección de archivos – El API Web Messaging – El JavaScript en segundo plano – El API WebSocket – El API de diseño

Luc VAN LANCKER

En los comienzos de Internet, **Luc VAN LANCKER**, entusiasmado con la idea de la comunicación universal que transmitía este concepto, se dedicó en cuerpo y alma a este ámbito. Es un formador apasionado, en contacto directo con las nuevas tecnologías relacionadas con la Web y un gran pedagogo. En Ediciones ENI, es autor de libros sobre HTML 4 y en diferentes colecciones, así como de libros sobre CSS, XHTML, jQuery, etc.

Introducción

Aquellos que vean Html5 como una sencilla actualización de Html 4.0 están equivocados. Html5 es una evolución considerable que modifica y modificará totalmente el diseño de las páginas o aplicaciones Web. Los fabricantes de navegadores, ya sean de escritorio o para Smartphone y tabletas, son totalmente conscientes de esto, porque han integrado rápidamente el concepto de Html5 y siguen de cerca sus evoluciones.

El concepto de Html5 se ha convertido en un trío ya que, además de las hojas de estilo CSS, se ha oficializado JavaScript como un socio de pleno derecho. Hay muchos API JavaScript que se han construido y hay otras muchas en desarrollo. Es muy importante observar que estos API JavaScript se implantan de manera nativa, lo que garantiza una ejecución rápida y, como consecuencia, una compatibilidad entre los diferentes navegadores. Y todo esto sin necesidad de llamadas a plug-ins o técnicas externas. Los lectores más experimentados descubrirán funcionalidades que ya existían en frameworks JavaScript, pero la mayor parte de estos API JavaScript son verdaderamente novedosos.

Este libro explora estos numerosos API JavaScript. Algunos ya están plenamente operativos en los navegadores recientes, mientras que otros todavía están en fase de desarrollo, incluso experimental y, como consecuencia, tendrán que evolucionar en el futuro.

Repasemos rápidamente estas nuevas funcionalidades de Html5, presentes en estos API. Los frameworks JavaScript y jQuery, en particular, habían incidido en las lagunas de JavaScript tradicional en la selección de los elementos del DOM. El API Selectors ha remediado estas carencias en gran medida. Probablemente, la novedad más mediática en la actualidad sea la geolocalización, que permite de conocer las coordenadas geográficas del usuario. Ahora, gracias al API Storage, es posible conservar datos en el navegador. Estos datos se podrán utilizar más adelante sin pasar por un servidor. Mejor incluso, se podrá tener una base de datos en el navegador. A partir de ahora es posible hacer editable para el usuario cualquier elemento de la página. Esto, junto con la posibilidad de almacenamiento de datos en local, aumentará todavía más la interactividad de las páginas o aplicaciones. Html5 es una tecnología abierta a Web Móvil que, con los Smartphone y otras tabletas, juega un papel cada vez más importante en la Web. Se ha construido un API para que sea posible seguir utilizando una aplicación en modo desconectado, después de una pérdida de conexión con la red. El uso masivo de Ajax ha perjudicado el uso del historial de páginas visitadas. El API History remedia esto, permitiendo crear nuevas entradas en este historial. La funcionalidad de drag&drop es otro de los clásicos de los frameworks JavaScript que se ha retomado de manera nativa en esta ocasión. Después tenemos una serie de API más limitados, como la selección de archivos, la posibilidad de transmitir información entre diferentes ventanas o etiquetas <iframe> del mismo dominio o de dominios diferentes, la ejecución de scripts en segundo plano y el API WebSocket, que permite tener una conexión bidireccional permanente entre el cliente y el servidor. Terminamos por el API Canvas, que permite el diseño 2D directamente en la página sin tener que utilizar imágenes.

Este estudio es innovador y confuso al mismo tiempo. Hemos tratado de adoptar un enfoque pragmático y explicativo, ilustrado con numerosos ejemplos y capturas de pantalla.

El objetivo de este libro es doble. En primer lugar, permite al lector integrar en sus aplicaciones alguno de estos API, como la geolocalización, el diseño en 2D, el almacenamiento de datos en local o, por qué no, una base de datos. En segundo lugar, permite descubrir el enorme impulso que van a crear estos API JavaScript que serán, en su conjunto, una plataforma completa de desarrollo de aplicaciones Html5.

Html5

A medida que se descubre Html5, los desarrolladores se dan cuenta, cada vez más, de su enorme potencial. El concepto Html5 aporta más funcionalidades, innovaciones y mejoras en el rendimiento, que renovarán totalmente el paisaje de las aplicaciones Web en los próximos meses o años.

Html 3.2 permite crear páginas Web utilizando solo etiquetas y atributos del lenguaje Html. Html 4.0 asocia las hojas de estilo CSS al Html para el diseño gráfico de las páginas. Durante este periodo, aparecieron muchos frameworks JavaScript para cubrir las lagunas de este dúo Html + CSS, en lo que respecta a la interactividad y a las funcionalidades. De manera lógica, el concepto Html5 contó con JavaScript como nuevo socio. De esta manera, el diseño de páginas y aplicaciones Web con Html5 utiliza el trío Html5 + CSS + JavaScript.



Esto confirma que JavaScript se ha convertido en un socio de pleno derecho en el diseño de páginas Web. Hay que resaltar que su importancia ha ido aumentando desde su implementación en Netscape 2, en 1993. Desde sus inicios, casi anecdóticos, este sencillo lenguaje de script se vio reforzado por la aparición del DOM y la adopción masiva por parte de los desarrolladores de la tecnología Ajax. Los diseñadores de los navegadores Web han percibido este aumento de potencia en la tecnología y han renovado sus motores JavaScript.

De esta manera, Html5 ofrece de manera nativa, en un entorno integrado, API JavaScript estandarizados. Una vez que se han implementado en los navegadores, recientes o futuros, estos API JavaScript permiten independizarse de los lenguajes del lado servidor para convertirse, en el lado cliente, en completas plataformas de desarrollo.

Los navegadores se han unido a este nuevo impulso innovador en lugar de esperar, como sucedía en el pasado, a que existiera una norma oficial. Ya sea para las etiquetas Html5 o para las hojas de estilo CSS3, estos API JavaScript estandarizados se han implementado rápidamente en los navegadores recientes o lo serán en próximas versiones.

Los API JavaScript de Html5 de este libro

Los API JavaScript de Html5 tienen puntos fuertes en su diseño:

- Hasta hace poco tiempo, Html estaba reservado a los ordenadores de sobremesa. Con Html5 y los API JavaScript, los Smartphone y tabletas Web se han convertido rápidamente en actores activos y reconocidos de la Web. Algunos de estos API JavaScript están destinados claramente a Web Móvil.
- El acento se ha puesto en dotar de más funcionalidades a los navegadores en el lado cliente y, a medio y largo plazo, disminuir la carga de trabajo de los servidores muy solicitados por un uso, cada día en aumento, de la Web.
- Algunos de estos API ya se utilizaban ampliamente en los frameworks JavaScript como jQuery, Dojo, MooTools o Prototype, por citar solo algunos. Algunos API JavaScript han integrado de manera nativa algunas funcionalidades que eran bien conocidas por los desarrolladores experimentados gracias a estos frameworks.
- Frente a la riqueza de estos API JavaScript, se percibe el objetivo de reforzar el poder de las aplicaciones Web frente a las aplicaciones de escritorio.
- Con el trío Html5, CSS y JavaScript, el concepto de Html se convierte en una completa plataforma de desarrollo.

En primer lugar, vamos a detallar el amplio alcance de los API JavaScript que se estudian en este libro.

Es importante señalar que el desarrollo de estos API JavaScript se realiza en diferentes niveles. Alguno de ellos se hallan en una fase avanzada y directamente operativos, mientras que otros todavía están en desarrollo o en fase experimental.

El API Selectors

Respecto a los frameworks, especialmente jQuery, el JavaScript clásico solo disponía de las funciones getElementById, getElementsByName y getElementsByTagName. Estas operaciones eran muy limitadas para la selección de elementos en el documento. Este API Selectors cubre al fin esta laguna, ofreciendo de manera nativa funcionalidades avanzadas que integran los selectores CSS3 para acceder a cada elemento del DOM de la página.

La geolocalización

El API de geolocalización es seguramente el más mediático e intrigante de todos los nuevos API JavaScript. Ya sea para los Smartphone o los ordenadores de sobremesa, permite, con la autorización del usuario, conocer la localización (latitud y longitud) del usuario. Gracias a esta localización, los sitios y las aplicaciones Web pueden responder mejor a las necesidades de información del usuario.

El almacenamiento de datos en local

Este API Storage permite almacenar datos en el navegador (es decir, en el lado cliente) en lugar de tener que llamar al servidor. Estos datos se pueden conservar durante la sesión del navegador o de manera permanente. Evitando las llamadas clásicas al servidor de aplicaciones, la navegación es más fluida, especialmente en el entorno Web móvil. Sin duda, este API tendrá repercusiones importantes en el diseño de las páginas o aplicaciones Web.

Una base de datos en el lado cliente (Web SQL Database)

Una base de datos en el lado cliente, gestionada por el navegador y no por el servidor, es otra de las novedades ofrecidas por los API JavaScript de Html5. Esta base de datos administrada por SQLite, que fue implantada en algunos navegadores de escritorio o Smartphone, ha quedado obsoleta a causa de las bases de datos Indexed Database. Sin embargo, este proceso podría perdurar todavía algunos años.

Una base de datos en el lado cliente (Indexed Database)

Este API retoma el mismo principio de base de datos del lado cliente, pero esta vez administrada por JavaScript, sin hacer llamadas a una tecnología externa (SQLite), con el objetivo de asegurar más

coherencia en la tecnología aplicada. Este API ilustra bien el aspecto experimental de los API JavaScript de Html5, ya que en la actualidad no está implementado en todos los navegadores.

La edición de contenido (contentEditable)

JavaScript es un actor fundamental en la interactividad de las páginas Web. Hasta ahora, la interacción de los usuarios tenía que pasar por campos de texto de los formularios. Con el atributo contentEditable, cualquier elemento de una página se puede editar. Este contenido editable, junto con el API de almacenamiento en local, vendrá a reforzar considerablemente esta interactividad de JavaScript. La evolución esperada de este API consiste en ofrecer al usuario las funcionalidades de un software de tratamiento de texto.

Las aplicaciones en modo desconectado (offline)

Este API offline está destinado especialmente a los Smartphone, ya que permite continuar la consulta de la página o utilizar una aplicación, sin una conexión momentánea de red. Es una mejora indiscutible que el diseñador puede aportar a su aplicación para hacerla operativa en modo *offline*. No obstante, el cacheo de la aplicación se distingue del proceso clásico de los navegadores, ya que el desarrollador puede decidir libremente los archivos que se pueden consultar sin conexión.

Manipular el historial del navegador

Con la tecnología Ajax y su amplia difusión, el historial y la preciada funcionalidad que permite volver a la página anterior que se ha consultado se ha visto perjudicada porque el contenido actual de la página ya no está relacionado con su URL. El objetivo de este API es solucionar este inconveniente tan importante. A partir de ahora es posible modificar de manera dinámica la URL en la barra de dirección del navegador o crear nuevas entradas en el historial sin recargar la página.

El drag&drop (arrastrar y soltar)

El drag&drop permite al usuario mover los elementos de la página. Esta interacción del usuario añade un efecto gráfico agradable. Sobre todo cuando con nuestros móviles y tabletas táctiles mover los elementos por la pantalla se ha convertido en algo habitual.

Este drag&drop es bien conocido por los usuarios de frameworks JavaScript como jQuery, Dojo, MooTools y otros. Este API ofrece una alternativa estandarizada e implementada de manera nativa en el navegador.

La gestión de archivos

El objetivo del API File es modernizar y extender las funcionalidades del antiguo campo de formulario <input type="file">, que existe desde los inicios del lenguaje Html y cuya eficacia no siempre estaba asegurada.

Este API permite seleccionar archivos desde el sistema operativo del usuario con la etiqueta <input type="file"> o, de manera más moderna, mediante un drag&drop aplicado al archivo seleccionado. Además, proporciona información útil del archivo gestionado, como el nombre, tamaño y formato. El objetivo final de este API, todavía en pleno desarrollo, será la gestión del proceso de *upload* asíncrono de estos archivos.

El API Web Messaging

El API Html5 Web Messaging permite transmitir cadenas de caracteres (*string*) entre diferentes ventanas o etiquetas <iframe>. Estas últimas pueden estar en el mismo dominio o -y es aquí donde está la novedad- en dominios diferentes.

El JavaScript en segundo plano

Con la llegada de los nuevos motores JavaScript en todos los navegadores, JavaScript puede enfrentarse a operaciones más complejas y, por lo tanto, más largas de realizar. El avance de una página Web era secuencial y la ejecución de estos largos scripts podía bloquear durante mucho tiempo el uso de la página. Normalmente esto conseguía molestar bastante a los usuarios y hacer que se fueran a otro sitio Web. El API Web Workers permite hacer que los scripts se ejecuten en paralelo al avance normal de la página, sin perturbar al usuario.

Los WebSockets

Con la tecnología Ajax, una página Web se puede convertir en un trasiego incesante entre el navegador y el servidor usando, según las reglas del protocolo HTTP, un procedimiento de preguntas/respuestas fastidioso. La idea de los WebSockets es permitir abrir y dejar abierto durante un tiempo determinado esta comunicación entre el navegador y el servidor, para ganar de esta manera un tiempo precioso. Esto permitirá imaginar aplicaciones en tiempo real, como, por ejemplo, las cotizaciones de las bolsas internacionales. Este API todavía está en una fase experimental, ya que los primeros ensayos han mostrado problemas importantes de seguridad. Pero el reto es de envergadura porque todo esto podría terminar con otro protocolo diferente al HTTP.

El API de diseño

Este API está muy extendido en los navegadores recientes. El API Canvas permite incluir gráficos directamente en la página usando código JavaScript sin tener que utilizar imágenes. Es un API potente, pero que aún tiene que mostrar su enorme potencial, ya que los diseñadores y otros expertos gráficos todavía lo han integrado poco.

Los futuros API JavaScript

Hay otros API JavaScript en desarrollo o en fase de proyecto. Veamos una lista que no pretende ser exhaustiva.

Media Capture API

Este API permite acceder a las capacidades del periférico de captura de audio, imagen y vídeo.

Html Media Capture

Esta especificación define las mejoras destinadas a los formularios Html para dotarlos de acceso a las capacidades del periférico de audio, foto o vídeo.

Server-Sent Events

Este API permitirá abrir una conexión HTTP para recibir las notificaciones *push* desde un servidor, en forma de eventos DOM.

Battery Status Event Specification

Esta especificación define un nuevo tipo de evento DOM que proporciona información sobre el estado de la batería de la unidad de alojamiento y de los dispositivos auxiliares asociados.

Touch Events Specification

Especificación de interfaz táctil que define un conjunto de eventos que representan uno o varios puntos de contacto con una superficie sensible al teclado, así como los cambios de estos puntos con relación a la superficie (por ejemplo, para las pantallas táctiles o las tabletas gráficas).

Messaging API

Este API permitirá acceder a las funcionalidades de mensajería electrónica, incluidos los SMS, MMS y correos electrónicos.

Filesystem API

Permitirá, cuando termine su desarrollo, organizar un sistema de directorios y archivos como en las aplicaciones de escritorio. Se podrán crear, leer y eliminar directorios y archivos. También se podrá copiar, renombrar y duplicar los directorios y archivos de este sistema.

Notifications Web

Un API para visualizar las notificaciones sencillas que se expiden al usuario fuera de la página actual, como la llegada de nuevos correos electrónicos o información procedente de una agenda.

Clipboard API and Events

Este API está destinado a la gestión del portapapeles, como copiar, cortar y pegar en las aplicaciones Web.

DeviceOrientation API

Esta especificación define diferentes tipos nuevos de eventos DOM, que ofrecen información sobre la orientación física y el movimiento de un periférico. Este API está dirigido claramente a los periféricos de Web Móvil.

Para un catálogo exhaustivo de estos API JavaScript de Html5, puede consultar el sitio Web:<http://dret.typepad.com/dretblog/html5-api-overview.html>

Los requisitos previos

Este libro está dirigido a los expertos y aspirantes a expertos en el dominio del diseño de aplicaciones Web. Los diversos API JavaScript de Html5 que se estudian en esta obra representan un verdadero avance en el dominio del desarrollo Web, dando al navegador funcionalidades innovadoras que formarán parte, sin ninguna duda, del paisaje de los futuros desarrollos Web.

Este libro exige una buena base en Html5 y CSS porque nos centraremos en ellos para abordar los ejemplos. A nivel de JavaScript, son necesarios conocimientos sólidos, sin que sea necesario ser un experto.

De hecho, cuando pasa el efecto desconcertante de la novedad, observará que se han hecho verdaderos esfuerzos en el código JavaScript de estos API para hacerlos asequibles y operativos.

Los navegadores de nuestro estudio

Google Chrome

Google se ha convertido en un actor omnipresente de la Web y nunca ha ocultado su interés por Html5, las hojas de estilo CSS3 y los nuevos API JavaScript. Sin reservas de clasificación entre los navegadores disponibles, Google Chrome es el navegador recomendado para el estudio de este libro. Todos los API JavaScript de este libro se hallan integrados en Google Chrome, incluso aquellos que todavía están en fase de desarrollo o experimental.

Las herramientas de desarrollo (ver más adelante en este capítulo) parecen diseñadas especialmente para ilustrar el resultado de los diferentes API que jalona esta obra.

El autor no concibe un estudio fructífero de los API JavaScript de Html5 sin tener a su disposición una versión reciente de Google Chrome.

Firefox

Firefox sigue siendo un excelente navegador, incluso cuando el autor, como otros usuarios, se muestra algo descontento por la cantidad de versiones. Firefox ha pasado de la versión 5 a la 11 en pocos meses, con mejoras poco significativas para el usuario lambda.

Firefox integra un buen número de API JavaScript de Html5, incluso en algunos casos es el promotor (por ejemplo, las bases de datos Indexed Database). El único inconveniente es que las herramientas de desarrollo como Firebug parecen haberse retrasado con relación a las herramientas de desarrollo que ofrecen los navegadores dotados del motor de renderizado webkit.

Internet Explorer

Internet Explorer 9 todavía no ha integrado muchos de los API JavaScript que se estudian en este libro. Baste con decir que las versiones 6, 7 y 8 no le serán de ninguna utilidad para este estudio. Microsoft promete que la versión de Internet Explorer 10 (que salió al mercado en junio de 2012) cubrirá esta laguna. Pero, así como Internet Explorer 9 excluye a los usuarios de XP y solo está disponible a partir de Windows Vista y 7, la versión 10 solo está disponible para los usuarios de Windows 7, lo que limita mucho la difusión de los API JavaScript de Html5 para los incondicionales de este navegador.

Sin embargo, Microsoft ha confirmado su interés por Html5 y JavaScript a través de la interfaz Metro de Windows 8, que está basada casi exclusivamente en estas dos tecnologías.

Safari (para Windows)

Safari es el navegador de referencia de la familia Macintosh. Dispone del mismo motor de renderizado (webkit) que Google Chrome y también está a la vanguardia de la integración de Html5 y sus nuevos API JavaScript. Los usuarios de Macintosh encontrarán en él un excelente socio para el estudio de este libro y el desarrollo de sus propios ejemplos.

Opera

Excelente navegador (si bien su difusión sigue siendo una incógnita), Opera siempre es innovador en todo lo que respecta a la integración de Html5 y sus nuevas etiquetas. No podemos ignorarlo, aunque solo aparece de manera puntual en nuestro estudio.

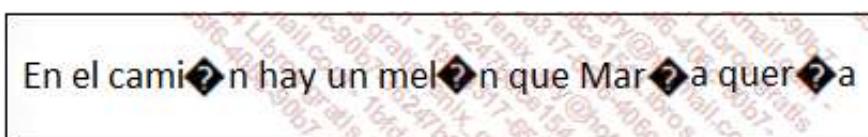
Las herramientas para nuestro estudio

En primer lugar, necesita un sencillo editor de texto. Cada sistema operativo proporciona uno; tanto que su uso es habitual, incluso obligatorio, para algunas tareas informáticas básicas. El autor es un incondicional de Notepad de Windows, incluso cuando hay otro software como Notepad++ o Notepad2 que también sirven para nuestro propósito. Los usuarios de Macintosh disponen de SimpleText o TextEdit, según la versión de su Mac OS.

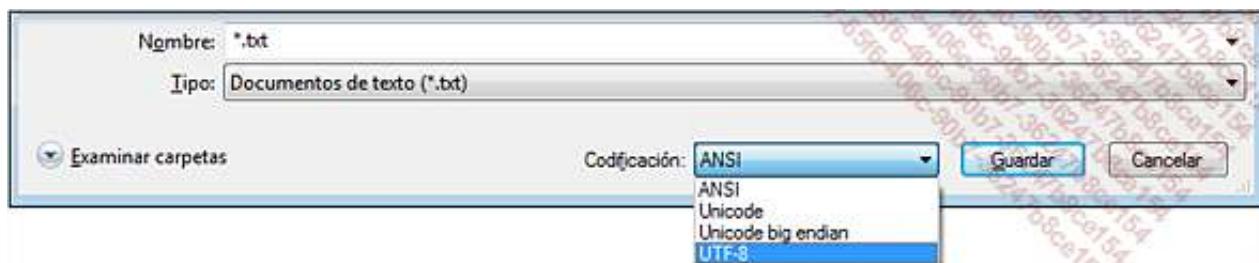
Lo importante es tener un editor de texto sin ningún formato.

W3C recomienda el juego de caracteres UTF-8. Así, todos nuestros ejemplos adoptarán esta forma de codificación de caracteres.

Si esta codificación no se especifica cuando se guarda el archivo Html, se corre el riesgo de desagradables sorpresas del tipo siguiente:



De esta manera, es imprescindible que tenga cuidado en codificar correctamente sus ejemplos en UTF-8.



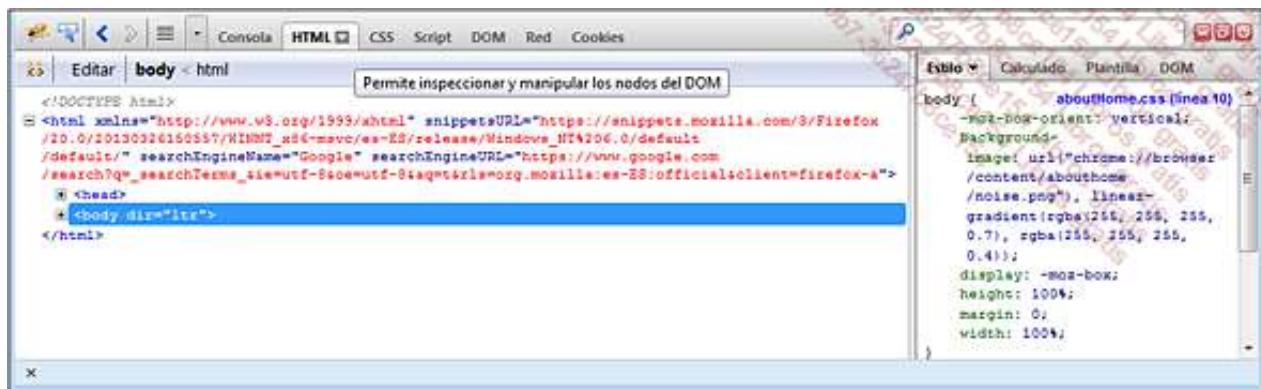
Este libro explora muchos API JavaScript de Html5. Algunos funcionan en local, como el API Canvas, y otros, como el API de geolocalización, necesitan un servidor Web local, como IIS (*Internet Information Services*) en Windows. Aquí, de nuevo, existen otras alternativas, como EasyPHP. Para terminar, otros API necesitarán una conexión para los ejemplos.

Nos aseguraremos de indicar esto al inicio de cada capítulo.

Las herramientas de desarrollo y depuración

JavaScript es un lenguaje de programación en el que incluso el desarrollador más experimentado comete errores de sintaxis. Para la depuración del JavaScript no existe una solución mágica, aunque ahora hay apreciables ayudas.

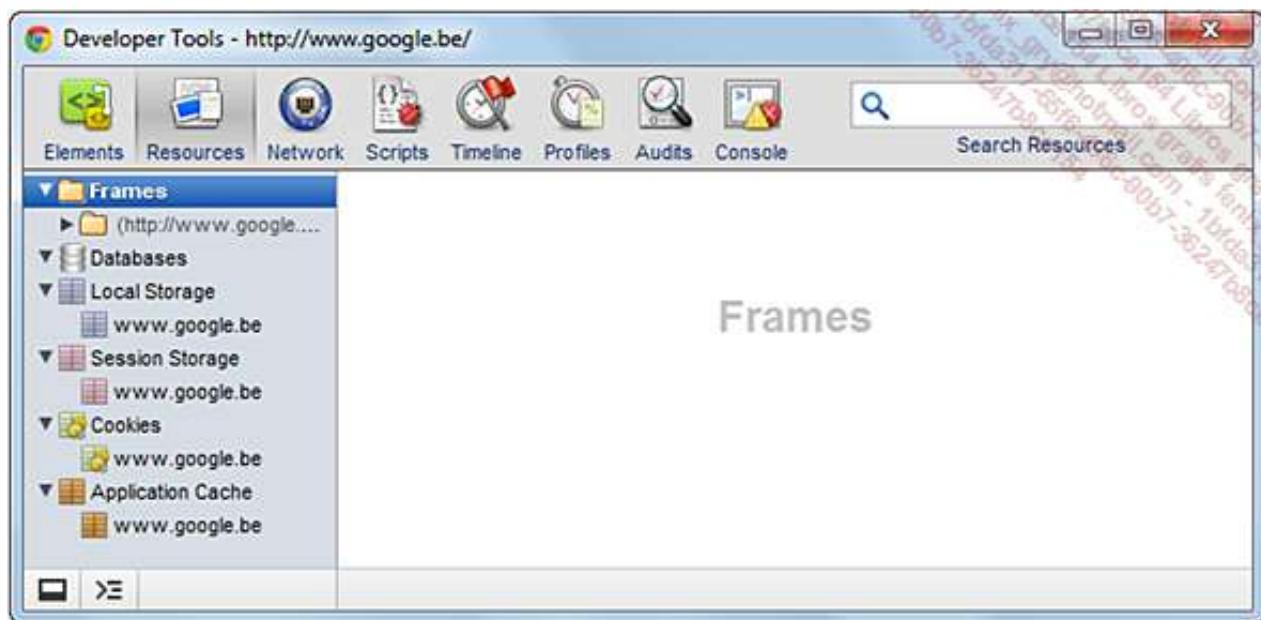
En primer lugar tenemos Firebug, una extensión de Firefox que permite controlar su HTML, CSS, JavaScript, DOM y Ajax.



Las herramientas de desarrollo de Google Chrome o Safari están perfectamente adaptadas a nuestro estudio y a los desarrollos basados en los API tratados en este libro.

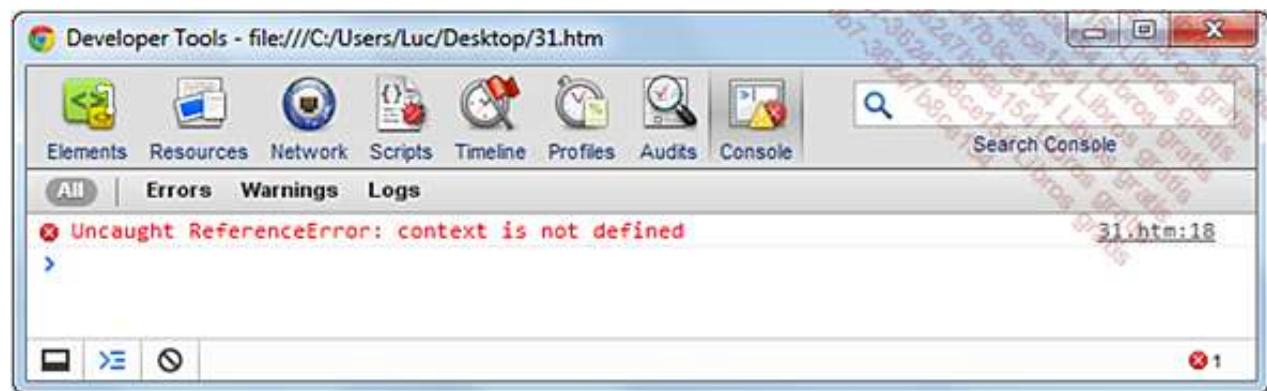
Para Google Chrome: Herramienta - **Herramientas para desarrolladores** o más rápidamente con el acceso directo [Ctrl][Shift] I.

Para Safari sobre Windows, en la barra de menú: **Desarrollo - Mostrar Web Inspector** o el acceso directo [Ctrl][Alt] I.



La pestaña **Resources** nos interesa particularmente. Observe las reglas **Databases** (bases de datos del lado cliente), **Local Storage** y **Session Storage** (datos conservados de manera permanente o durante la sesión), **Cookies** y **Application Cache**, que serán de gran ayuda para ilustrar nuestros propósitos y confirmar la correcta ejecución de los ejemplos de este libro.

Estas herramientas de desarrollo también le pueden ayudar en la depuración de los scripts (pestaña **Console** o **Scripts**).



Presentación del API

El JavaScript tradicional solo ofrece tres métodos para acceder a los elementos del DOM:`getElementById()`,`getElementsByName()` y`getElementsByTagName()`.

Los frameworks JavaScript, como jQuery, Prototype, Dojo o Moo Tools, se han centrado en esta carencia para sacar partido totalmente a los selectores CSS2 y sobre todo CSS3.

W3C ha desarrollado el API Selectors para ofrecer un concepto estandarizado que permita acceder de forma fácil a los elementos del DOM y beneficiarse completamente de los selectores CSS y también de sus estándares.

Por lo tanto, este nuevo API entra en competencia directa con las facilidades aportadas por los frameworks JavaScript en lo que respecta a la selección de elementos. Y pensamos en particular en jQuery, una de cuyas especialidades es la selección de los elementos del DOM.

Decir implementación de manera nativa es decir rapidez de ejecución. El sitio webkit.org ofrece una comparativa del API Selectors y los diferentes frameworks; los resultados son muy instructivos (www.webkit.org/perf/slickspeed/). A continuación mostramos un breve resumen para las pruebas realizadas 250 veces.

Selector CSS	API Selectors	jQuery
div	5 ms	19 ms
div div	6 ms	145 ms
div > div	6 ms	55 ms
div + div	7 ms	58 ms
div ~ div	7 ms	85 ms

Los ejemplos, pruebas y otros desarrollos de este API Selectors se pueden hacer en local.

Disponibilidad del API

Este API es recocido por los siguientes navegadores de escritorio:

- Internet Explorer 8+ pero solo los selectores CSS2.
- Internet Explorer 9+ para todos los selectores CSS, incluidos los CSS3.
- Firefox 3.6+.
- Google Chrome 1.0+.
- Safari 3.2+.
- Opera 10+.

Para los Smartphone y otras tabletas:

- iOS Safari 3.2+.
- Opera Móvil 10+.
- Android Móvil 2.1.

Podemos considerar este API Selectors igual de compatible con los navegadores recientes, porque solo los execrables Internet Explorer 6 e Internet Explorer 7 no lo soportan.

Siempre es posible probar la disponibilidad verificando el reconocimiento de document.querySelector o el navegador. Es decir:

```
if(document.querySelector) {  
// APS Selectors soportado  
}
```

En la web encontramos:

```
if (typeof document.querySelector != "function") {  
// El API Selectors no está soportado  
}
```

Ejemplo

```
<!DOCTYPE html>  
<html lang="es">  
<head>  
<title>Los API JavaScript de Html5</title>  
<meta charset=utf-8>  
<style type="text/css">  
#box { width: 270px;  
        border: 1px solid black;  
        background-color: rgb(195,215,235);  
        padding-left: 3px;  
        text-align: center; }  
</style>  
<script type="text/javascript">  
function init(){  
if(document.querySelector) {  
msg = "El API Selectors está soportado";  
document.querySelector('#box').innerHTML = msg;  
}  
else {  
alert("El API Selectors no está soportado");  
}  
}  
</script>
```

```
</head>
<body onload="init();">
<h2>API Selectors</h2>
<div id="box">&nbsp;</div>
</body>
</html>
```



Los selectores CSS

Veamos algunos selectores CSS.

Los selectores principales:

#identificador	Selecciona un elemento identificado por un id. #box	1
.clase	Selecciona los elementos que tienen una clase concreta. .box	1
etiqueta	Selecciona todas las etiquetas designadas. div	1
*	Selecciona todos los elementos. *	2

Los selectores jerárquicos:

A, B	Selecciona todos los elementos A y todos los elementos B. div, p	1
A B	Selecciona todo elemento B que es descendiente de un elemento A. ul li	1
A > B	Selecciona todo elemento B que es hijo directo de un elemento A. div > span	2
A + B	Selecciona todo elemento B que está inmediatamente precedido de un elemento A. ul#lista li	2
A ~ B	Selecciona todo elemento B precedido de un elemento A. ul ~ li	3

Las pseudoclases de selección:

:empty	Corresponde a los elementos vacíos y que no tienen hijos. div:empty	3
:focus	Corresponde al elemento que tiene el foco. input:focus	2
:checked	Corresponde a los elementos marcados en los formularios de radio y casillas de selección. input [type=checkbox] :checked	3
:enabled	Selecciona todos los elementos input que están activados. input:enabled	3
:disabled	Selecciona todos los elementos input que están desactivados. input:disabled	3
:only-child	Selecciona el hijo único. div:only-child	3
:first-child	Selecciona el primer hijo.	2

	<code>ul:first-child</code>	
<code>:last-child</code>	Selecciona el último hijo. <code>ul:last-child</code>	3
<code>:nth-child(n)</code>	Designa al enésimo elemento hijo. <code>ul:nth-child(3)</code>	3
<code>:nth-child(even)</code>	Designa los elementos hijos pares. <code>ul:nth-child(even)</code>	3
<code>:nth-child(odd)</code>	Designa los elementos hijos impares. <code>ul:nth-child(odd)</code>	3
<code>:nth-last-child(n)</code>	Designa el enésimo elemento hijo partiendo del último elemento. <code>ul:nth-last-child(2)</code>	3
<code>:first-of-type</code>	Representa el primer elemento de este tipo. <code>li:first-of-type</code>	3
<code>:last-of-type</code>	Representa el último elemento de este tipo. <code>li:last-of-type</code>	3
<code>:nth-of-type(n)</code>	Representa el enésimo elemento de este tipo. <code>li:nth-of-type(3)</code>	3
<code>:nth-last-of-type(n)</code>	Representa el enésimo elemento de este tipo partiendo del último elemento. <code>li:nth-last-of-type(2)</code>	3
<code>:not(selector)</code>	Recupera todos los elementos que no responden a las condiciones del selector. <code>:not (p)</code>	3
<code>:contains(valor)</code>	Recupera los elementos para los que el contenido textual contiene la subcadena dada como argumento. <code>p:contains ('Nombre')</code>	3
<code>::selection</code>	Aplica la propiedad de estilo a la selección del texto de elemento que ha hecho el usuario. <code>p::selection</code>	3
<code>::before</code>	Genera un contenido antes de un elemento. <code>p::before</code> Existe en CSS2 con el formato :before.	3
<code>::after</code>	Genera un contenido después de un elemento. <code>p::after</code> Existe en CSS2 con el formato :after.	3

Los selectores de atributos:

<code>[attr]</code>	Designa al elemento que tiene el atributo attr que se indica. <code>[class]</code>	2
<code>[attr="valor"]</code>	Designa un elemento que tiene el atributo attr proporcionado por el valor que se indica. <code>[class="clase1"]</code>	2
<code>[attr~="valor"]</code>	Corresponde a todo elemento cuyo atributo attr contiene una lista de valores separados por un espacio y en el que uno de ellos es valor.	2

	[class~="clase1"]	
[attr^="valor"]	Representa un elemento cuyo atributo attr empieza con el prefijo dado por el valor que se indica. [class^="cl"]	3
[attr\$="valor"]	Representa un elemento cuyo atributo attr termina por el sufijo dado por el valor que se indica. [class\$="1"]	3
[attr*="valor"]	Representa un elemento cuyo atributo attr contiene una instancia del valor que se indica. [class*= "las"]	3

Seleccionar por el nombre de una clase CSS (getElementsByClassName)

HTML5 ha oficializado el selector `getElementsByClassName` que permite acceder a un elemento en función del nombre de la clase a la que está asociado. Esto, que ya existía desde hace mucho tiempo en algunos navegadores (por ejemplo, Firefox 3), ahora es un estándar de W3C. Observe que este selector no está soportado por Internet Explorer hasta su versión 9.

La sintaxis es:

```
document.getElementsByClassName(nombre);
```

donde `nombre` es una cadena de caracteres que representa el nombre de la clase de los elementos que se quiere seleccionar.

Ejemplo, para seleccionar todos los elementos que tienen la clase `test`:

```
document.getElementsByClassName('test')
```

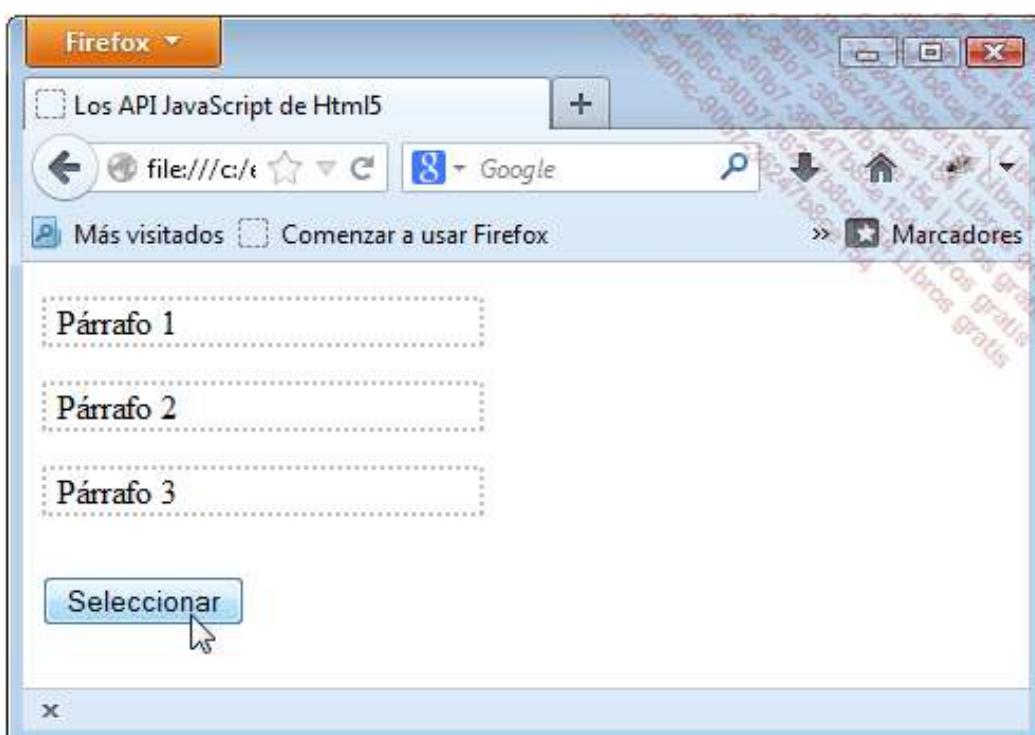
Es importante observar que el valor devuelto es una variable `Array`, ya que puede haber varios elementos que cumplan la condición de selección.



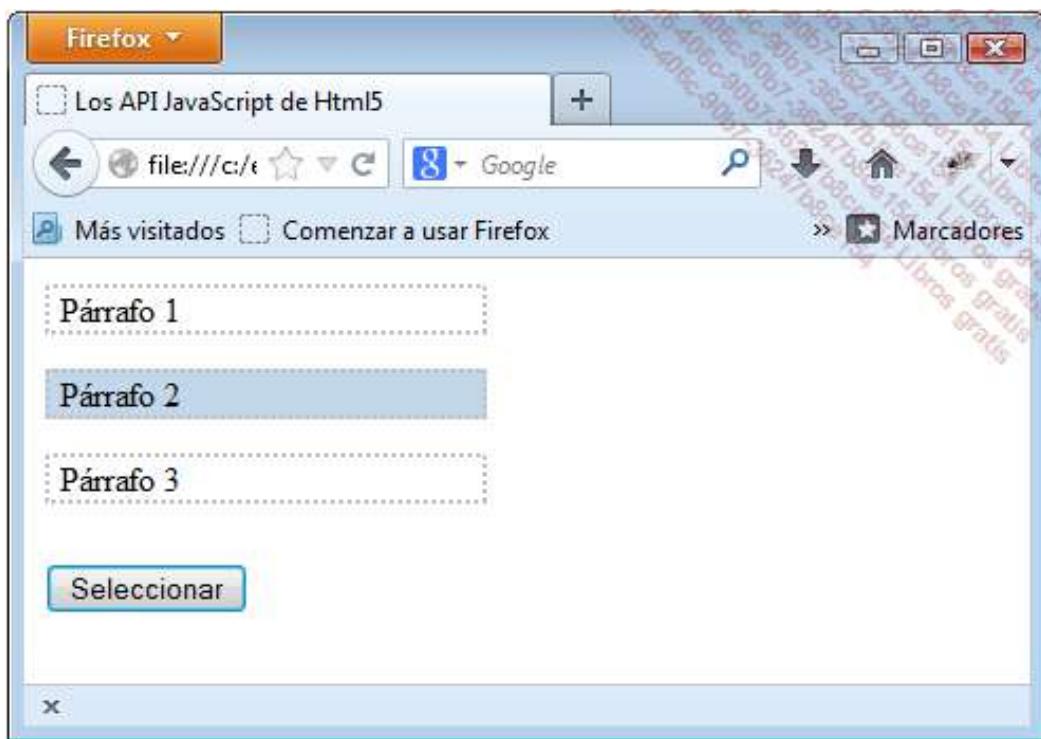
Ejemplo 1

Consideremos tres párrafos. Vamos a seleccionar aquellos que tienen el atributo de clase `dos`.

Cuando ejecutamos el archivo:



Al hacer clic en el botón, el párrafo se selecciona:



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
p { width: 200px;
    border: 2px dotted silver;
    padding-left: 5px; }
button { margin-top: 12px; }
</style>
<script type="text/javascript">
function go() {
var x = document.getElementsByClassName("dos");
x[0].style.background = "rgb(195,215,235)";
}
</script>
</head>
<body>
<p class="uno"> Párrafo 1</p>
<p class="dos"> Párrafo 2</p>
<p class="tres"> Párrafo 3</p>
<button onclick="go ()">Seleccionar</button>
</body>
</html>
```

Comentario

```
var x = document.getElementsByClassName("dos");
```

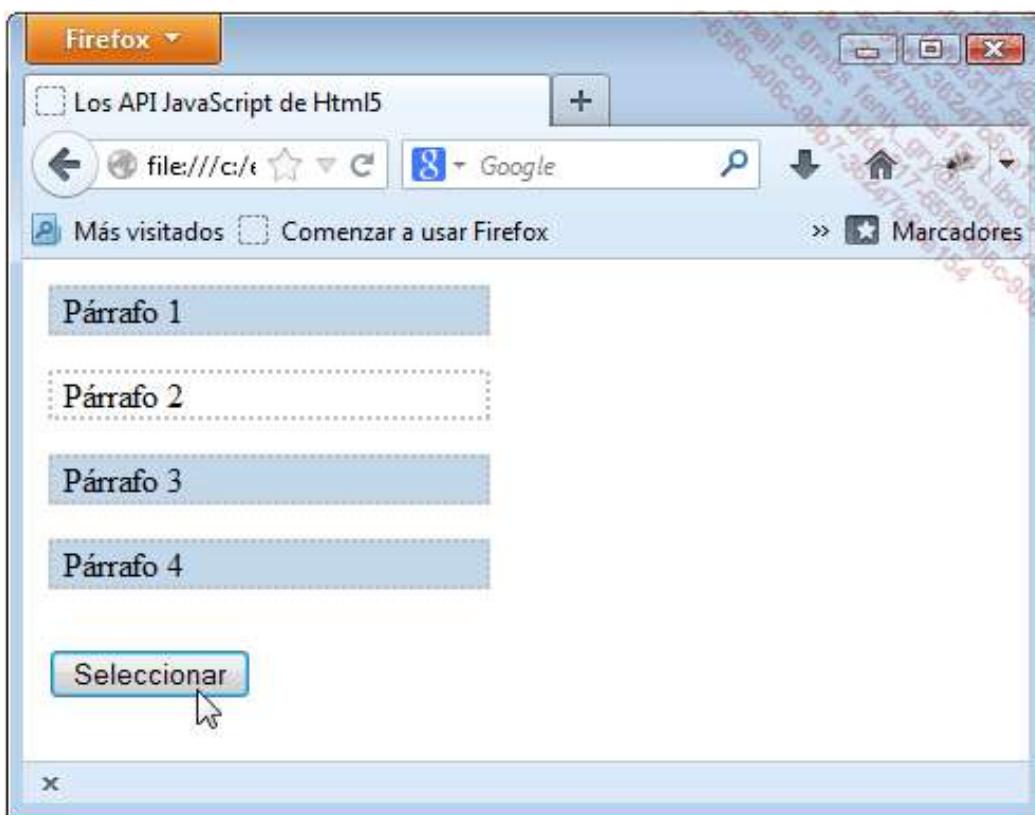
La variable `x` contiene los elementos con un atributo `class="dos"`. En este caso solo hay un único elemento, pero podría haber varios. Por eso, la variable `x` es de tipo Array.

```
x[0].style.background = "rgb(195,215,235)";
```

A este único elemento seleccionado (`x[0]`), se le aplica un color de fondo.

Ejemplo 2

Retomemos el ejemplo anterior, pero esta vez con varios elementos con la misma clase.



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
p { width: 200px;
    border: 2px dotted silver;
    padding-left: 5px; }
button { margin-top: 12px; }
</style>
<script type="text/javascript">
function go() {
var x = document.getElementsByClassName("box");
for (i = 0; i < x.length; i++) {
x[i].style.background = "rgb(195,215,235)";
}
}
</script>
</head>
<body>
<p class="box"> Párrafo 1</p>
<p class="otra"> Párrafo 2</p>
<p class="box"> Párrafo 3</p>
<p class="box"> Párrafo 4</p>
<button onclick="go()">Seleccionar</button>
</body>
</html>
```

Comentario

```
var x = document.getElementsByClassName("box");

getElementsByClassName selecciona varios párrafos con la clase class="box"
getElementsByClassName.

for (i = 0; i < x.length; i++) {
x[i].style.background = "rgb(195,215,235)";
}
```

Hay que tratar en un bucle `for` estos diferentes elementos para añadirles un color de fondo.

Seleccionar el primer elemento de un conjunto (querySelector)

El método `document.querySelector()` devuelve el primer elemento encontrado que corresponde al selector. La sintaxis es:

```
document.querySelector(selector)
```

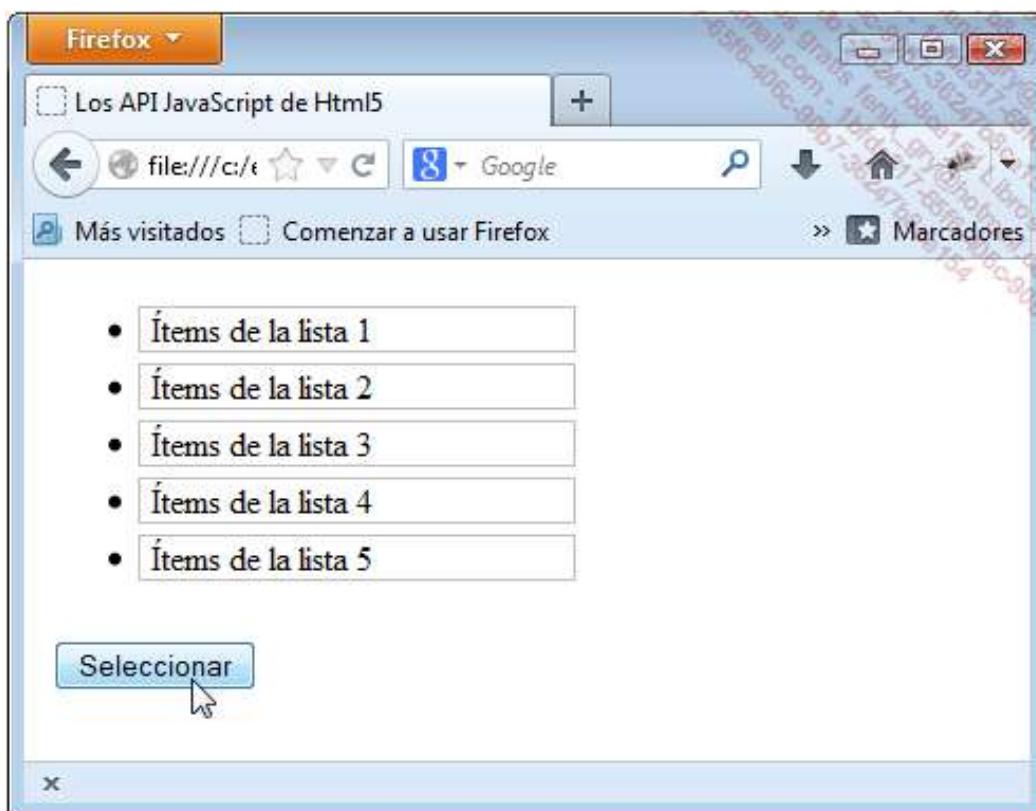
De esta manera, en lugar de `document.getElementById("contenido")`, es equivalente usar `document.querySelector("#contenido")`. Observe la presencia de la almohadilla (#) para `querySelector()`.

El método `querySelector()` solo selecciona un elemento, lo que revela rápidamente sus límites.

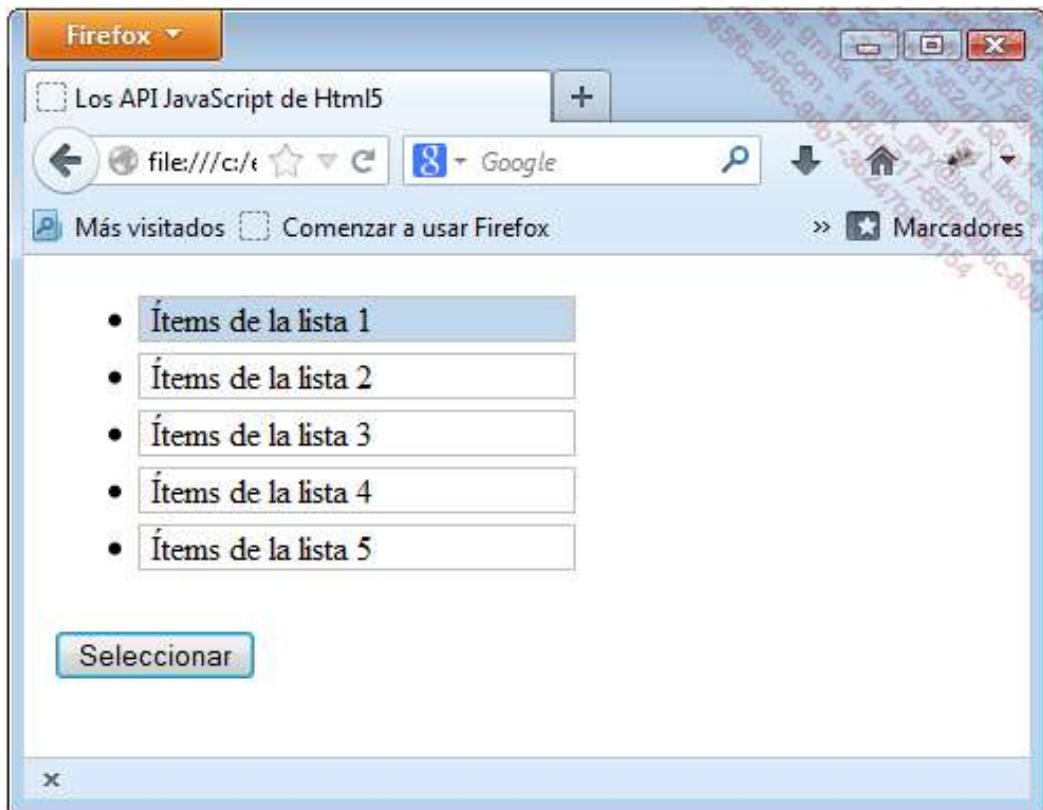
Ejemplo 1

Vamos a seleccionar el primer ítem de una lista.

Al cargar la página:



Después de hacer clic en el botón:



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
li { width: 200px;
    border: 1px solid silver;
    margin-bottom: 5px;
    padding-left: 5px; }
button { margin-top: 12px; }
</style>
<script type="text/javascript">
function go() {
document.querySelector('li' ).style.background =
"rgb(195,215,235)";
}
</script>
</head>
<body>
<ul>
<li>Ítems de la lista 1</li>
<li>Ítems de la lista 2</li>
<li>Ítems de la lista 3</li>
<li>Ítems de la lista 4</li>
<li>Ítems de la lista 5</li>
</ul>
<button onclick="go ()">Seleccionar</button>
</body>
</html>
```

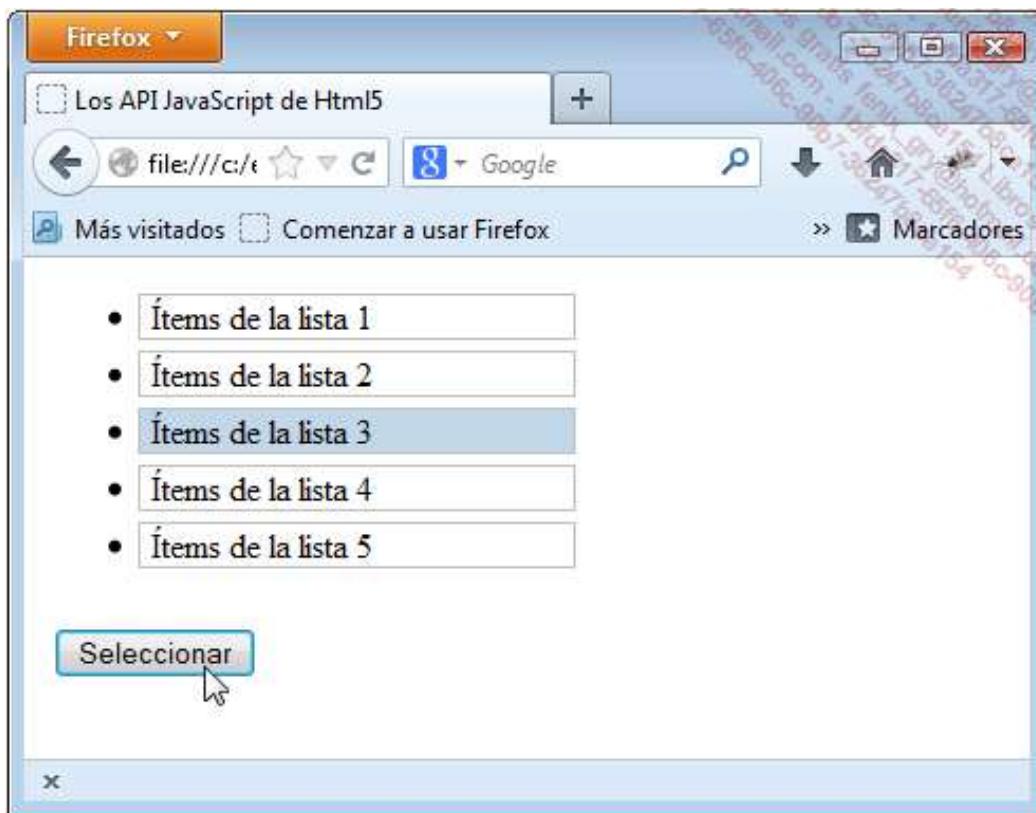
Comentario

```
document.querySelector('li' ).style.background = "rgb(195,215,235)";
```

`querySelector('li')` selecciona el primer elemento `` y se le aplica un color de fondo.

Ejemplo 2

Siempre con nuestra lista, seleccionamos el tercer ítem. Aprovecharemos la ocasión para usar un selector CSS3.



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
li { width: 200px;
    border: 1px solid silver;
    margin-bottom: 5px;
    padding-left: 5px; }
button { margin-top: 12px; }
</style>
<script type="text/javascript">
function go() {
document.querySelector("li:nth-of-type(3)").style.background =
"rgb(195,215,235)"; }
</script>
</head>
<body>
<ul>
<li>Ítems de la lista 1</li>
<li>Ítems de la lista 2</li>
<li>Ítems de la lista 3</li>
<li>Ítems de la lista 4</li>
<li>Ítems de la lista 5</li>
</ul>
<button onclick="go()">Seleccionar</button>
</body>
```

```
</html>
```

Comentario

```
document.querySelector("li:nth-of-type(3)").style.background = "rgb(195,215,235)";
```

Se selecciona el tercer elemento de la lista ("li:nth-of-type(3)") y se le aplica un color de fondo.

Ejemplo 3

Vamos a seleccionar un elemento de lista vacío.



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
li { width: 200px;
    border: 1px solid silver;
    margin-bottom: 5px;
    padding-left: 5px; }
button { margin-top: 12px; }
</style>
<script type="text/javascript">
function go() {
var x = document.querySelector("li:empty").style.background =
"rgb(195,215,235)";
}
</script>
</head>
<body>
<ul>
<li>Ítems de la lista 1</li>
```

```
<li>ítems de la lista 2</li>
<li></li>
<li>ítems de la lista 4</li>
<li>ítems de la lista 5</li>
</ul>
<button onclick="go()">Seleccionar</button>
</body>
</html>
```

Comentario

```
var x = document.querySelector("li:empty").style.background = "rgb(195,215,235)";
```

El selector CSS3 ("li:empty") encuentra el ítem de lista vacío.

Seleccionar un conjunto de elementos (querySelectorAll)

No es extraño que, como consecuencia de la manipulación del DOM, tratemos al mismo tiempo varios elementos. Es aquí donde interviene el método `querySelectorAll()`, que devuelve todos los elementos que corresponden al selector.

```
document.querySelectorAll(selector)
```

Ejemplo

`document.querySelectorAll('p')` devuelve todos los párrafos `<p>` del documento.

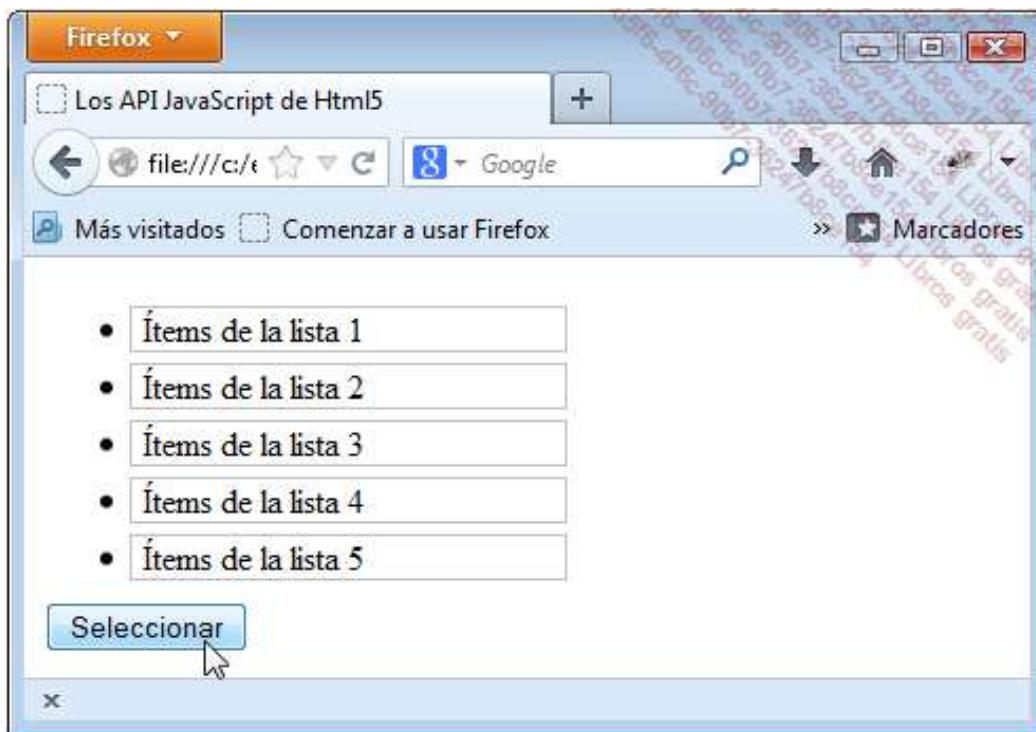
Como sucede con el método `getElementsByClassName` del punto 2, todos los elementos devueltos forman una variable Array que se tratará en el script con la ayuda de un bucle `for`.

`querySelectorAll` también existe en jQuery desde la versión 1.4.3.

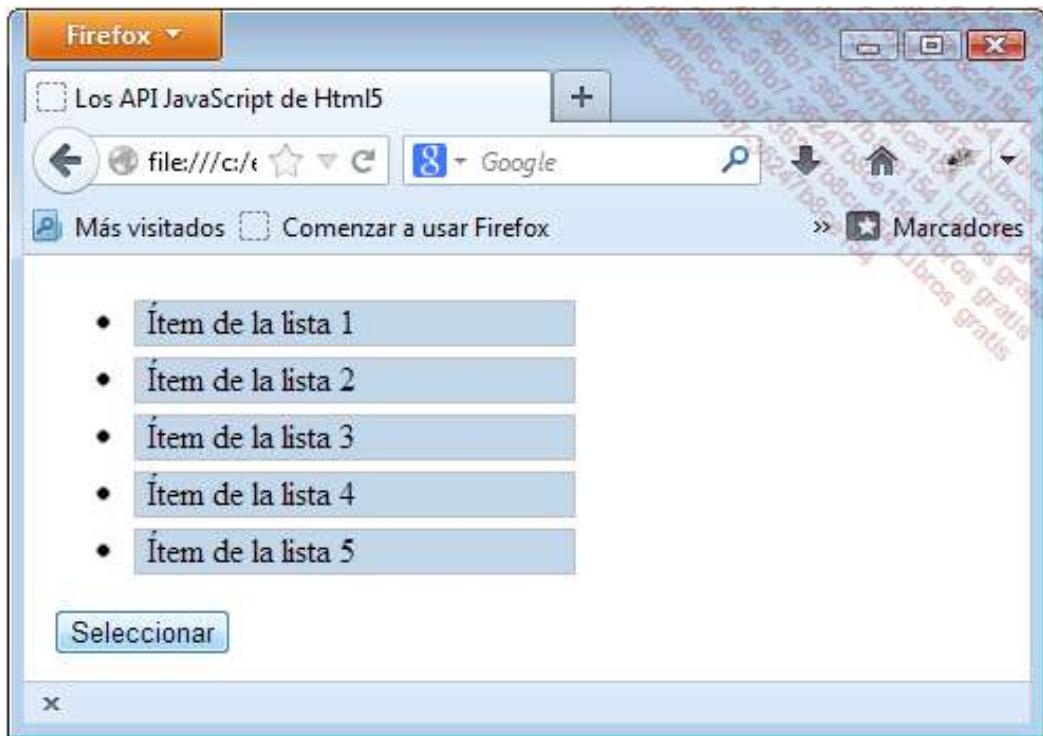
Ejemplo 1

Retomemos nuestro ejemplo 1 del punto anterior, pero en esta ocasión le aplicamos el método `querySelectorAll`.

Al cargar la página:



Después hacer clic en el botón **Seleccionar**:



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
li { width: 200px;
    border: 1px solid silver;
    margin-bottom: 5px;
    padding-left: 5px; }
button { margin-top: 12px; }
</style>
<script type="text/javascript">
function go() {
items=document.querySelectorAll('li' );
for (i = 0; i < items.length; i++) {
items[i].style.background = "rgb(195,215,235)";
}
}
</script>
</head>
<body>
<ul>
<li>Ítems de la lista 1</li>
<li>Ítems de la lista 2</li>
<li>Ítems de la lista 3</li>
<li>Ítems de la lista 4</li>
<li>Ítems de la lista 5</li>
</ul>
<button onclick="go ()">Seleccionar</button>
</body>
</html>
```

Comentario

```
items=document.querySelectorAll('li' );
```

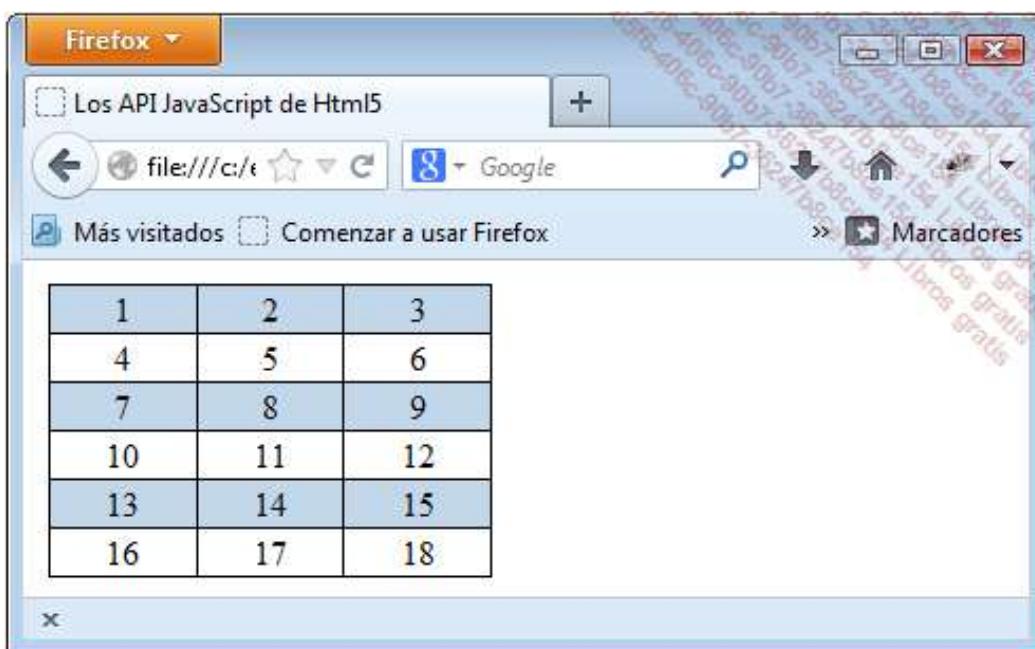
La variable `items` contiene todos (`querySelectorAll`) los ítems de la lista.

```
for (i = 0; i < items.length; i++) {
  items[i].style.background = "rgb(195,215,235)";
}
```

Un bucle `for` recorre los elementos de la variable `items` y les aplica un color de fondo (`style.background = "rgb(195,215,235)"`). Al final, todos los ítems de la lista están coloreados.

Ejemplo 2

Vamos a aplicar un efecto *listing* a una matriz.



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
table { width: 210px;
        border-collapse: collapse;
        border: 1px solid black; }
td { text-align: center;
     border: 1px solid black; }
</style>
<script type="text/javascript">
function go() {
var lineas = document.querySelectorAll("table tr:nth-child(odd)");
for (i = 0; i < lineas.length; i++) {
lineas[i].style.background = "rgb(195,215,235)";
}
}
</script>
</head>
<body onload="go()">
<table>
<tr><td>1</td><td>2</td><td>3</td></tr>
<tr><td>4</td><td>5</td><td>6</td></tr>
<tr><td>7</td><td>8</td><td>9</td></tr>
<tr><td>10</td><td>11</td><td>12</td></tr>
<tr><td>13</td><td>14</td><td>15</td></tr>
<tr><td>16</td><td>17</td><td>18</td></tr>
</table>
```

```

<tr><td>7</td><td>8</td><td>9</td></tr>
<tr><td>10</td><td>11</td><td>12</td></tr>
<tr><td>13</td><td>14</td><td>15</td></tr>
<tr><td>16</td><td>17</td><td>18</td></tr>
</table>
</body>
</html>

```

Comentario

```
var lineas = document.querySelectorAll("table tr:nth-child(odd)");
```

El script selecciona todos los elementos de línea <tr> impares (odd) en el orden de los hijos de la etiqueta <table>.

```

for (i = 0; i < lineas.length; i++) {
lineas[i].style.background = "rgb(195,215,235)";
}

```

Un bucle for recorre las líneas seleccionadas y les aplica un color de fondo (style.background = "rgb(195,215,235)").

No puedo resistirme a tratar este ejemplo en jQuery, que presenta una escritura más concisa, evitando el fastidioso bucle for.

```

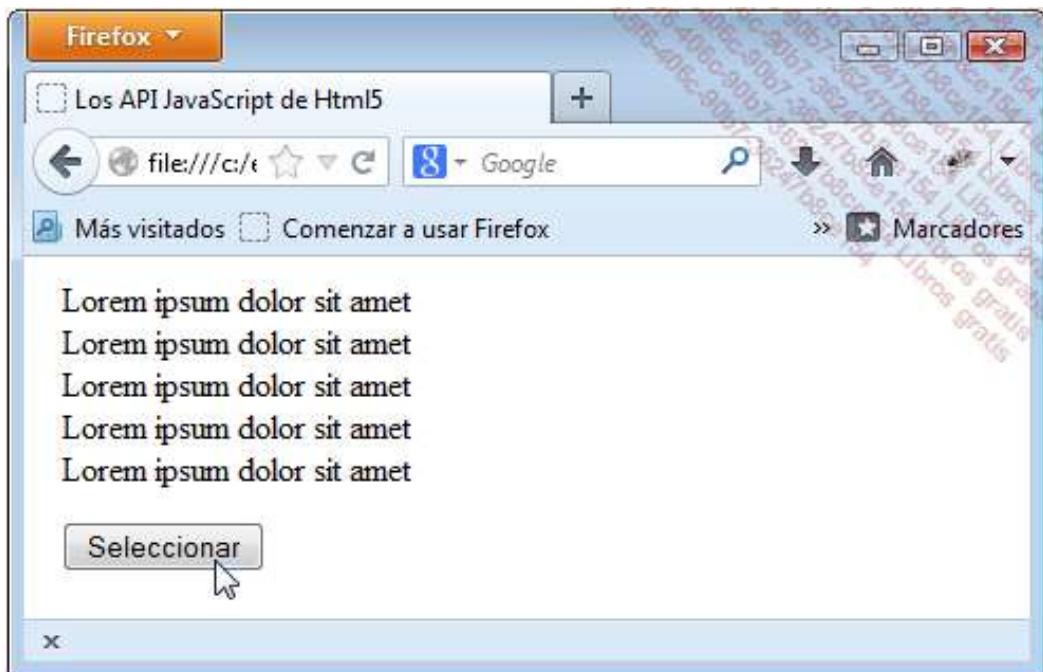
<!DOCTYPE html>
<html lang="es">
<head>
<title>jQuery</title>
<meta charset=utf-8>
<style type="text/css">
table { width: 210px;
        border-collapse: collapse;
        border: 1px solid black; }
td { text-align: center;
     border: 1px solid black; }
</style>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.6/
jquery.min.js"></script>
<script type="text/javascript">
$(document).ready(function(){
$("table tr:nth-child(odd)").css("background",
"rgb(195,215,235)");
});
</script>
</head>
<body>
<table>
<tr><td>1</td><td>2</td><td>3</td></tr>
<tr><td>4</td><td>5</td><td>6</td></tr>
<tr><td>7</td><td>8</td><td>9</td></tr>
<tr><td>10</td><td>11</td><td>12</td></tr>
<tr><td>13</td><td>14</td><td>15</td></tr>
<tr><td>16</td><td>17</td><td>18</td></tr>
</table>
</body>
</html>

```

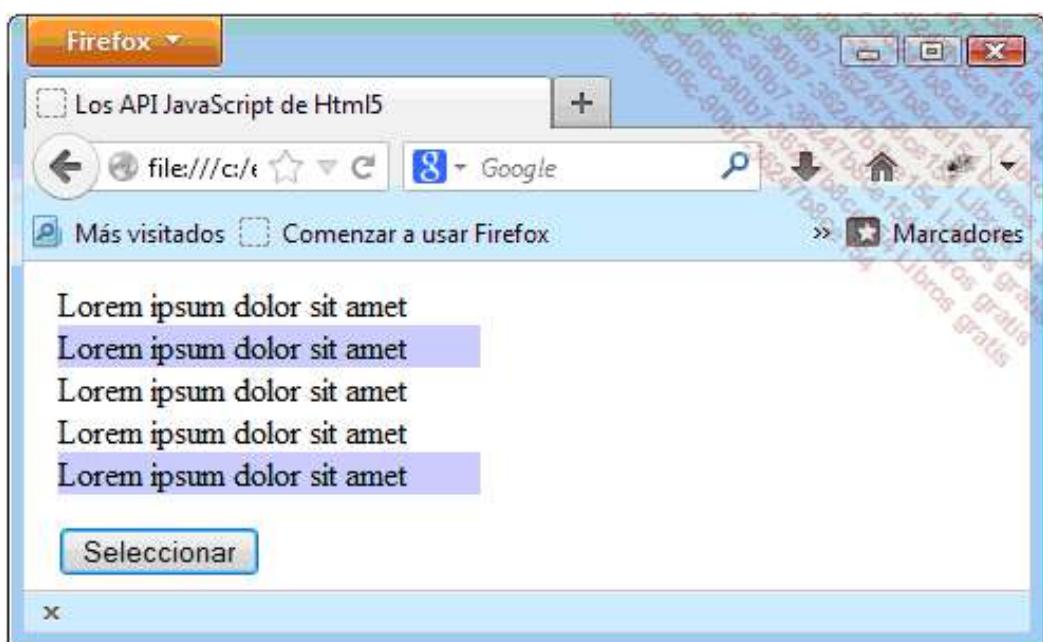
Ejemplo 3

Consideremos varias capas con atributos variados. Accedemos a los atributos name en cuyo contenido aparezcan las letras im-.

Al inicio:



El resultado:



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>jQuery</title>
<meta charset=utf-8>
<style type="text/css">
div { width: 200px; }
button { margin-top: 15px; }
</style>
<script type="text/javascript">
function go() {
atributos=document.querySelectorAll('div[name*="im-"]');
for (i = 0; i < atributos.length; i++) {
atributos[i].style.background = "rgb(195,215,235)";
```

```

}
}
</script>
</head>
<body>
<div class=" antepenultimo">Lorem ipsum dolor sit amet</div>
<div name="prim-ero">Lorem ipsum dolor sit amet</div>
<div name="dos">Lorem ipsum dolor sit amet</div>
<div title=" penultimo">Lorem ipsum dolor sit amet</div>
<div name="ultim-o">Lorem ipsum dolor sit amet</div>
<button onclick="go()">Seleccionar</button>
</body>
</html>

```

Comentario

```
atributos=document.querySelectorAll('div[name*="im-"]');
```

El selector de atributo CSS3 '`div[name*="im-"]`' selecciona las capas con el atributo `name`, cuyo contenido incluye las letras `im-`. Buen ejemplo del grado de sofisticación de los selectores CSS3.

```
for (i = 0; i < atributos.length; i++) {
    atributos[i].style.background = "rgb(195,215,235)";
}
```

Un bucle `for` aporta un efecto de estilo a los diferentes elementos.

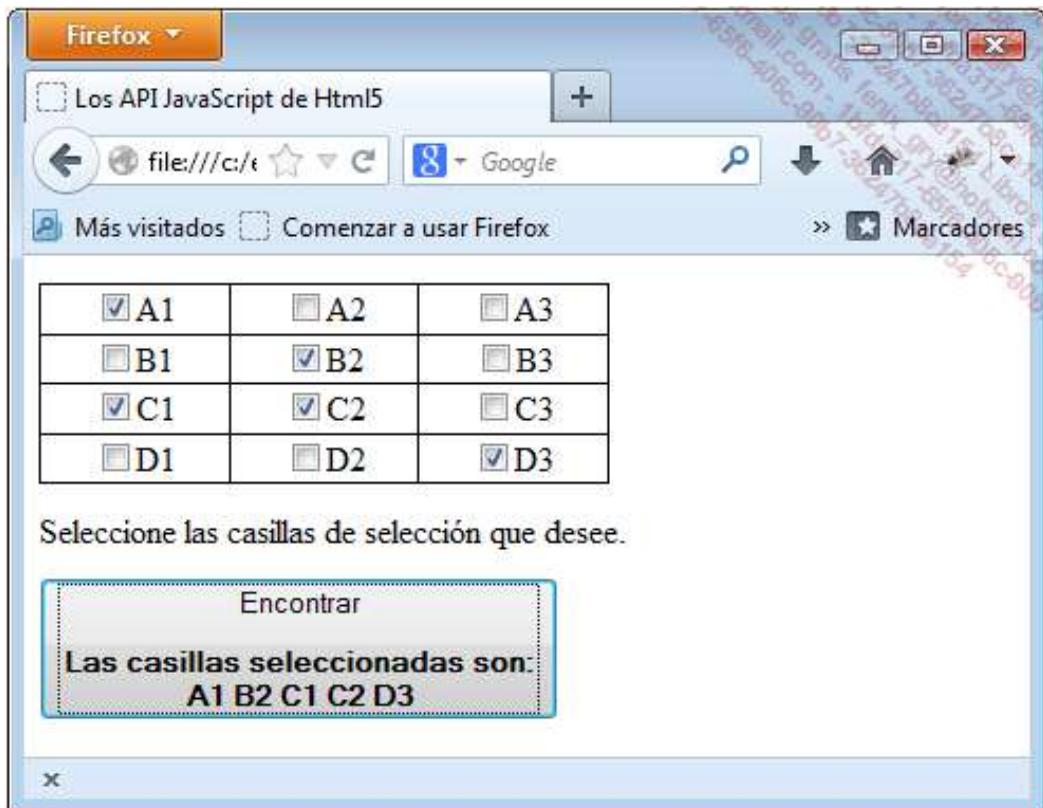
Ejemplo 4

Listemos las casillas marcadas en una matriz.

La situación inicial es la siguiente:



Al final:



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>jQuery</title>
<meta charset=utf-8>
<style type="text/css">
table { width: 270px;
        border-collapse: collapse;
        border: 1px solid black;
        margin-top: 15px;}
td { text-align: center;
    border: 1px solid black;}
#resultado { font-weight:bold; }
div { margin-top: 12px; }
button { margin-top: 12px; }
</style>
<script type="text/javascript">
function init() {
document.querySelector("#findChecked").onclick = function() {
var selected = document.querySelectorAll(":checked");
var result = "Las casillas seleccionadas son:<br> ";
for (var i = 0; i < selected.length; i++) {
result += (selected[i].name + " ");
}
document.querySelector("#resultado").innerHTML = result;
}
}
</script>
</head>
<body onload="init()">
<section>
<table>
<tr>
<td><input type="checkbox" name="A1">A1</td>
<td><input type="checkbox" name="A2">A2</td>
```

```

<td><input type="checkbox" name="A3">A3</td>
</tr>
<tr>
<td><input type="checkbox" name="B1">B1</td>
<td><input type="checkbox" checked="" name="B2">B2</td>
<td><input type="checkbox" name="B3">B3</td>
</tr>
<tr>
<td><input type="checkbox" name="C1">C1</td>
<td><input type="checkbox" name="C2">C2</td>
<td><input type="checkbox" name="C3">C3</td>
</tr>
<tr>
<td><input type="checkbox" name="D1">D1</td>
<td><input type="checkbox" name="D2">D2</td>
<td><input type="checkbox" name="D3">D3</td>
</tr>
</table>
<div>Seleccione las casillas de selección que desee.</div>
<button type="button" id="findChecked" autofocus="">Encontrar
</button>
<div id="resultado"></div>
</section>
</body>
</html>

```

Comentario

```

document.querySelector("#findChecked").onclick = function() {
...
}

```

Cuando se hace clic (onclick) en el botón (querySelector ("#findChecked")).

```
var selected = document.querySelectorAll(":checked");
```

La variable selected contiene todas (querySelectorAll) las casillas marcadas ("*:checked").

```

for (var i = 0; i < selected.length; i++) {
result += (selected[i].name + " ");
}
document.querySelector("#resultado").innerHTML = result;
}
```

El bucle for recorre los elementos de la variable selected. La variable result, que mostrará los elementos marcados, va a buscar el valor del atributo name asociado a la etiqueta <input>. Entonces se incluye en la capa identificada por resultado (querySelector ("#resultado")).

Presentación y objetivos

La geolocalización (*geolocation*) es un procedimiento que permite conocer las coordenadas geográficas de una persona y si es preciso transportarlas a un mapa o plano. A nivel de la Web, el API de geolocalización permite a un sitio Web conocer las coordenadas geográficas (longitud y latitud) de un usuario en la Web.

Este concepto no es nuevo en la Red. La novedad reside en el hecho de que esta funcionalidad se incluye de manera nativa en Html5, sin librería o API adicionales.

Si conoce la posición del usuario, un sitio Web puede responder de manera más precisa a sus necesidades. Las aplicaciones de geolocalización pueden ser múltiples. Un sitio Web comercial puede ofrecer su mercancía a los distribuidores cercanos al lugar en el que reside el usuario. Una red social le puede indicar otros usuarios que correspondan a su perfil y que residan cerca de usted. Un motor de búsqueda también le puede ofrecer hoteles, restaurantes o lugares de ocio en los alrededores de una ciudad concreta. En función de su lugar de residencia, un sitio Web de venta *online* podrá mostrar los precios en su moneda o las fechas en su formato habitual (dd/mm/aaaa o mm/dd/aaaa). Pero no hay que dejarse engañar: el objetivo principal de la geolocalización es fundamentalmente la publicidad y el poder dirigir mejor los mensajes al público objetivo en función de su posición geográfica.

¿De dónde provienen estos datos que permiten localizarle?

Para los Smartphone, se pueden utilizar dos procedimientos:

- La triangulación respecto a tres antenas.
- El GPS si su móvil o su tableta dispone de esta función. En este caso la precisión es casi total, aunque este método consume mucha energía e implica un tiempo de inicialización relativamente largo.

Para los ordenadores de escritorio, la geolocalización se basa en:

- La dirección IP del usuario. Se trata de un procedimiento en el que es sencillo conocer la dirección IP del usuario: basta con consultar los registros de asignación para conocer su dirección física. En ocasiones la precisión de la geolocalización está sujeta a sorpresas, ya que en algunos casos se devuelve la dirección de su proveedor de acceso.
- Su red Wi-Fi (dirección MAC). La geolocalización de su red Wi-Fi se obtiene por triangulación respecto a los puntos de acceso o terminales Wi-Fi cercanos. Recuerde que en mayo de 2010 Google reveló que los vehículos encargados de tomar fotos para su aplicación Google Street View también recogían información relativa a las redes Wi-Fi que se encontraban. La geolocalización que se obtenía de esta manera era muy precisa, sobre todo en las zonas urbanas.

Los ejemplos se deben ejecutar en un servidor Web local.

Disponibilidad del API

Para los ordenadores de escritorio:

- Internet Explorer 9.0+.
- Firefox 3.6+.
- Safari 5+.
- Chrome 5+.
- Opera 10.6+.

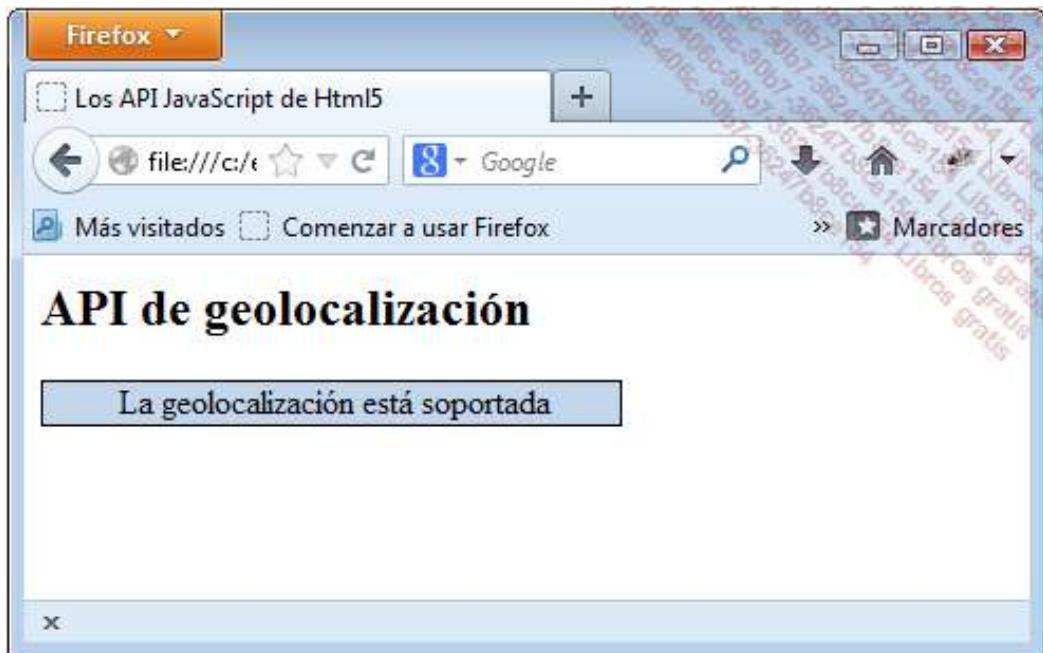
Para los Smartphone y tabletas:

- iOS Safari 3.2+.
- Opera Móvil 11+.
- Android Móvil 2.1+.

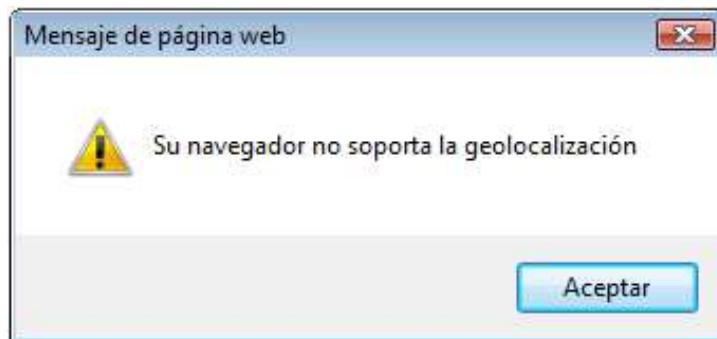
Para probar la disponibilidad para el usuario, es necesario saber que geolocation es un subobjeto del objeto navigator de JavaScript. De esta manera, navigator.geolocation va a enviar un sencillo booleano true o false en función de la disponibilidad de la geolocalización.

El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#box { width: 270px;
       border: 1px solid black;
       background-color: rgb(195,215,235) ;
       padding-left: 3px;
text-align: center; }
</style>
<script type="text/javascript">
function init(){
if(document.querySelector) {
msg = "La geolocalización está soportada";
document.querySelector('#box').innerHTML = msg;
}
else {
alert("Su navegador no soporta la geolocalización");
}
}
</script>
</head>
<body onload="init();">
<h2>API de geolocalización</h2>
<div id="box">&nbsp;</div>
</body>
</html>
```



En Internet Explorer 8:



Obtener la localización del usuario

La puesta en marcha de la geolocalización pasa por el siguiente método:

```
getCurrentPosition(función_en_caso_de Éxito, función_en_caso_de_error, opciones);
```

La función en caso de error y las opciones son argumentos opcionales.

En caso de éxito, es posible acceder al objeto position que devuelve las coordenadas (coords) y el momento en que se ha tomado la posición (time stamp).

El código JavaScript position.coords devuelve las coordenadas junto con otros valores:

- position.coords.longitude devuelve la longitud de la posición.
- position.coords.latitude devuelve la latitud de la posición.
- position.coords.altitude devuelve la altitud de la posición.
- position.coords.accuracy indica la precisión de las coordenadas.
- position.coords.altitudeAccuracy proporciona la precisión de la altitud.
- position.coords.heading devuelve la posición en grados respecto al norte. Observe que esto no es tan preciso como la brújula interna de un Smartphone.
- position.coords.speed corresponde a la velocidad del usuario en relación con su última posición.

Estos valores todavía no están presentes en los navegadores, pero dan una idea del potencial de esta aplicación.

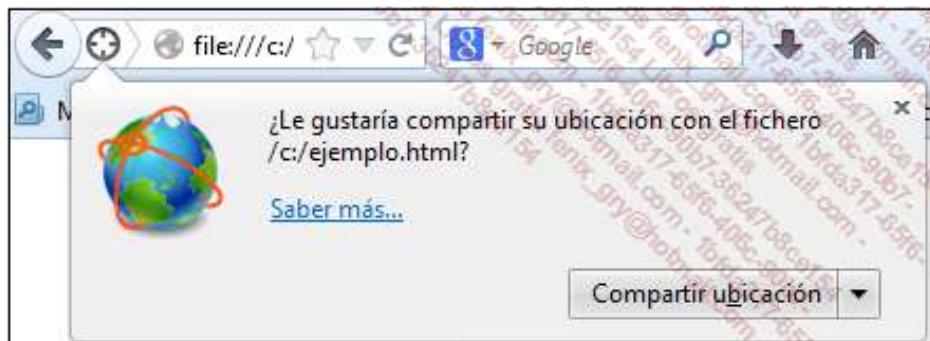
Ejemplo

Hagamos nuestra primera geolocalización.

Inicialmente:



Aparece un mensaje solicitando su autorización para ser localizado.



Al final:



Observe que las coordenadas se dan en formato decimal.

Para preservar los datos privados del autor, en el ejemplo anterior las coordenadas de longitud y de latitud corresponden a un lugar cualquiera.

El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style>
div { margin-bottom: 3px; }
span { border: 1px solid silver;
       padding-left: 4px;
       padding-right: 4px; }
#boton { margin-top: 12px;
          margin-left: 60px; }
</style>
<script type="text/javascript">
function localizar(position) {
var latitud = position.coords.latitude;
var longitud = position.coords.longitude;
document.getElementById("latitud").innerHTML = latitud;
document.getElementById("longitud").innerHTML = longitud;
}
function getLocalisation() {
```

```
if(navigator.geolocation){  
navigator.geolocation.getCurrentPosition(localizar);  
}  
else {  
alert("Lo sentimos, su navegador no soporta la  
geolocalización");  
}  
}  
</script>  
</head>  
<html>  
<body>  
<form>  
<h2>Html5 geolocalización</h2>  
<div>Latitud: <span id="latitud">En espera...</span></div>  
<div>Longitud: <span id="longitud">En espera...</span></div>  
<input type="button" id="boton" onclick="getLocalisation();"  
value="Geolocalizar">  
</form>  
</body>  
</html>
```

Administrar los errores de la localización

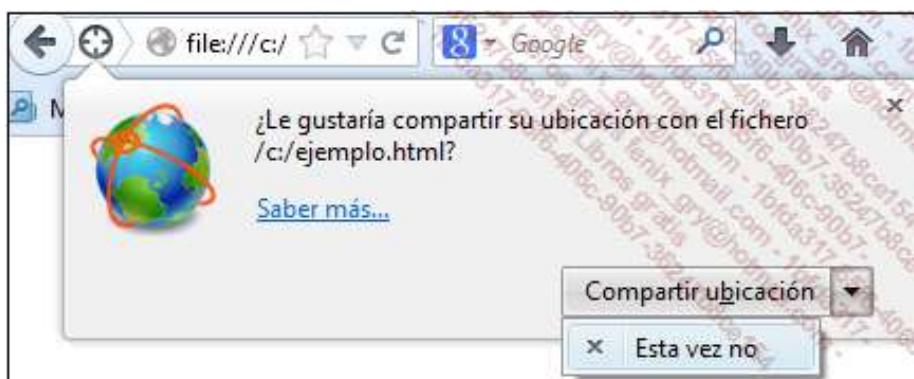
La función de gestión de errores tiene como argumento un objeto de tipo `PositionError`.

A continuación se da la lista de propiedades de este objeto:

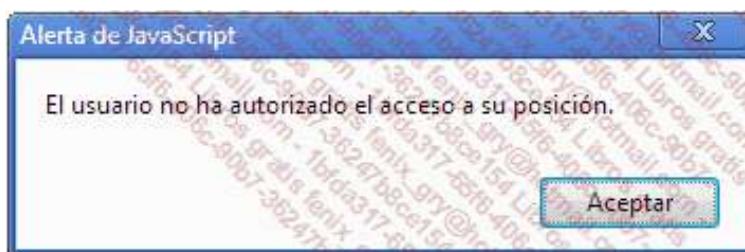
0	UNKNOWN_ERROR	Error desconocido.
1	PERMISSION_DENIED	El usuario no tiene autorización para acceder a su posición.
2	POSITION_UNAVAILABLE	La posición del usuario no es correcta.
3	TIMEOUT	El servicio no ha respondido en el tiempo límite.

Ejemplo

Utilizando el siguiente código, rechazamos la autorización a la página que hemos creado.



Se muestra una alerta anunciando que el usuario ha rechazado la autorización de geolocalización.



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style>
div { margin-bottom: 3px; }
span { border: 1px solid silver;
       padding-left: 4px;
       padding-right: 4px; }
#boton { margin-top: 12px;
          margin-left: 60px; }
</style>
<script type="text/javascript">
function localizar(position) {
var latitud = position.coords.latitude;
var longitud = position.coords.longitude;
```

```
document.getElementById("latitud").innerHTML = latitud;
document.getElementById("longitud").innerHTML = longitud;
}
function error(error) {
switch(error.code) {
case error.UNKNOWN_ERROR:
alert("La geolocalización ha encontrado un error.");
break;
case error.PERMISSION_DENIED:
alert("El usuario no ha autorizado el acceso a su posición.");
break;
case error.POSITION_UNAVAILABLE:
alert("El usuario no puede ser localizado.");
break;
case error.TIMEOUT:
alert("La geolocalización ha excedido el tiempo límite.");
break;
}
}
function getLocalisation(){
if(navigator.geolocation){
navigator.geolocation.getCurrentPosition(localizar,error);
}
else {
alert("Lo sentimos, su navegador no soporta la geolocalización");
}
}
</script>
</head>
<html>
<body>
<form>
<h2>Html5 geolocalización</h2>
<div>Latitude: <span id="latitud">En espera...</span></div>
<div>Longitude: <span id="longitud">En espera...</span></div>
<input type="button" id="boton" onclick="getLocalisation();"
value="Geolocalizar">
</form>
</body>
</html>
```

Opciones de la localización

Hay previstos argumentos avanzados para este API:

enableHighAccuracy	Permite una geolocalización más precisa. No tiene ninguna utilidad para los Smartphone que dispongan de la función GPS. Los valores son true o false (valor predeterminado). Esta opción consume muchos recursos, con lo que se corre el riesgo de agotar rápidamente la batería del periférico.
timeout	Expresa en milisegundos el tiempo de espera de la aplicación antes de hacer la llamada a la función de gestión de errores. El valor predeterminado es 0.
maximumAge	Determina en milisegundos el tiempo en que la posición actual está cacheada. Si el valor es 0 (valor predeterminado), la posición no se cacheará, sino que se renovará constantemente. También se prevé el valor infinito.

A nivel de código, estas opciones pueden tener la siguiente forma:

```
var opciones = {  
    enableHighAccuracy: true,  
    maximumAge: 60000,  
    timeout: 45000  
};  
navigator.geolocation.getCurrentPosition(exito, error, opciones);
```

o de manera más directa:

```
navigator.geolocation.getCurrentPosition(exito, error,  
{enableHighAccuracy: true, timeout:10000, maximumAge:600000});
```

[El código completo](#)

```
<!DOCTYPE html>  
<html lang="es">  
<head>  
<title>Los API JavaScript de Html5</title>  
<meta charset=utf-8>  
<style>  
div { margin-bottom: 3px; }  
span { border: 1px solid silver;  
       padding-left: 4px;  
       padding-right: 4px; }  
#boton { margin-top: 12px;  
         margin-left: 60px; }  
</style>  
<script type="text/javascript">  
function localizar(position) {  
    var latitud = position.coords.latitude;  
    var longitud = position.coords.longitude;  
    document.getElementById("latitud").innerHTML = latitud;  
    document.getElementById("longitud").innerHTML = longitud;  
}  
function error(error) {  
    switch(error.code) {  
        case error.UNKNOWN_ERROR:  
            alert("La geolocalización ha encontrado un error.");  
            break;  
        case error.PERMISSION_DENIED:  
    }
```

```
    alert("El usuario no ha autorizado el acceso a su posición.");
    break;
  case error.POSITION_UNAVAILABLE:
    alert("El usuario no se ha podido localizar.");
    break;
  case error.TIMEOUT:
    alert("La geolocalización ha excedido el tiempo límite.");
    break;
}
}

function getLocalisation(){
if(navigator.geolocation){
  navigator.geolocation.getCurrentPosition(localizar, error,
  {timeout:10000, maximumAge:600000});
}
else {
  alert("Lo sentimos, su navegador no soporta la geolocalización");
}
}
</script>
</head>
<html>
<body>
<form>
<h2>Html5 geolocalización</h2>
<div>Latitud: <span id="latitud">En espera...</span></div>
<div>Longitud: <span id="longitud">En espera...</span></div>
<input type="button" id="boton" onclick="getLocalisation();"
value="Geolocalizar">
</form>
</body>
</html>
```

Convertir las coordenadas decimales en hexadecimales

El API de geolocalización devuelve las coordenadas en formato decimal. Algunos prefieren expresar estas coordenadas de manera más clásica, en grados, minutos y segundos. A continuación se muestra un script para hacer esta conversión.

Al cargar la página:



El resultado de la conversión es:



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Convertidor</title>
<meta charset=utf-8>
<style type="text/css">
table { border: 1px solid silver;
         border-collapse: collapse; }
td { border: 1px solid silver;
     padding-left: 4px; }
#celda { width: 30%; }
</style>
<script type="text/javascript">
function valor_deg(v) {
var vv = "";
var correctos = "0123456789.";
var signo = 1;
var factor = 1;
var d = 0;
var c;
var oldc;
// Para los datos no numéricos
oldc = ' ';
for (i = 0; i < v.length; i++) {
var c = v.charAt(i).toUpperCase();
if (c == 'W' || c == 'S' || c == '-') {
signo = -1;
}
if (correctos.indexOf(c) < 0) {
c = ' ';
}
if (oldc != ' ' || c != ' ') {
vv += c;
oldc = c;
}
}
v = new Array();
v = vv.split(' ');
for (i = 0; i < v.length; i++) {
d += v[i] * factor;
factor /= 60;
}
return d * signo;
}
function redondear (v, p) {
return Math.round(v * Math.pow(10, p)) / Math.pow(10, p);
}
function convertir(v) {
var d, m, signo = '', str;
v = valor_deg(v);
if (v < 0) {
signo = '-';
v = - v;
}
str = signo + redondear (v, 6);
str += '<br>';
d = Math.floor(v);
v = (v - d) * 60;
str += signo + d.toString() + '&deg;' + redondear (v, 3) + "'";
str += '<br>';
m = Math.floor(v);
v = (v - m) * 60;
str += signo + d.toString() + '&deg;' + m.toString() + "' " +
```

```
redondear(v, 2) + "'";
id = document.getElementById('deg_out');
id.innerHTML = str;
}
</script>
</head>
<body>
<h2>Convertidor</h2>
<form name=d onsubmit="return false;">
<table>
<tr>
<td id="celda">Convertir</td>
<td>
<input name=d type="text" size="25"
onkeyup="convertir(this.value)">
</td>
</tr>
<tr><td>Resultado:</td>
<td><span id="deg_out">Grados<br>Grados Minutos<br>
Grados Minutos Segundos</span></td>
</tr>
</table>
</form>
</body>
</html>
```

Aplicación relacionada con Google Maps

Google Maps es el API que necesitamos si queremos ver la posición del usuario en un mapa, sobre todo porque es gratuito y muy sencillo de usar.

Normalmente hay que inscribirse en Google y solicitar una clave de identificación para usar su API, pero hay un truco. Es suficiente con añadir la siguiente línea.

```
<script type="text/javascript" src="http://maps.google.com/maps/api/js?sensor=true"></script>
```

Para nuestra aplicación serán suficientes algunos argumentos, pero para el resto de las opciones le invito a leer la documentación oficial de Google Maps:<http://code.google.com/intl/es-ES/apis/maps/documentation/javascript/basics.html>

Además de las líneas de script, hay que prever una ubicación para nuestro mapa, es decir, una capa en el cuerpo del código Html:

```
<div id="mapa"></div>
```

La creación de un nuevo mapa por Google Maps se hace simplemente con el siguiente código, que proporciona como argumentos la latitud y longitud encontrados por el API de geolocalización.

```
var latlng = new google.maps.LatLng(latitud, longitud);
```

Aprovechemos para explicar algunas opciones:

- El zoom (zoom). El nivel de zoom está incluido entre 0 y 21.
- El centrado del mapa (center). Normalmente retendremos la posición de la geolocalización en el centro del mapa.
- El tipo de mapa que ofrece Google Maps (mapTypeId). Usaremos ROADMAP para el plano, SATELLITE para las vistas aéreas, HYBRID para las vistas aéreas con el nombre de las calles y TERRAIN para ver los relieves.

El código de estas opciones puede tener la siguiente forma para un zoom de 15, centrando la localización y usando el modo mapa:

```
var myOptions = {  
    zoom: 15,  
    center: latlng,  
    mapTypeId: google.maps.MapTypeId.ROADMAP  
};
```

Para terminar, podemos ver el resultado, con las opciones previstas, en la capa central que se ha reservado para tal efecto.

```
var map = new google.maps.Map(document.getElementById("mapa"),  
myOptions);
```

Opcionalmente se puede prever un marcador para indicar la posición.

```
var marker = new google.maps.Marker({  
...  
});
```

Existe la posibilidad de utilizar algunas opciones, como la posición (position) o un texto que aparezca cuando se pase por encima del marcador (title):

Un ejemplo de código completo de marcador podría ser el siguiente:

```
var marker = new google.maps.Marker({  
    position: latlng,
```

```
map: map,
title:"Usted se encuentra aquí."
});
```

Ejemplo

Vamos a implementar un ejemplo que incluya todo esto.



Para preservar los datos privados del autor, las coordenadas de longitud y de latitud corresponden a un lugar cualquiera.

El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
</head>
<style type="text/css">
#map_canvas {
height: 300px;
width: 400px;
}
</style>
</head>
<script src="http://maps.google.com/maps/api/js?sensor=false">
</script>
<script type="text/javascript">
if (navigator.geolocation) {
navigator.geolocation.getCurrentPosition(ver_mapa, error);
}
else {
alert("Lo sentimos, su navegador no soporta la geolocalización");
}
```

```

}
// Success callback function
function ver_mapa(pos) {
var latitud = pos.coords.latitude;
var longitud= pos.coords.longitude;
var thediv = document.getElementById('localisationinfo');
thediv.innerHTML = '<p>Su longitud es <b>' + longitude + '</b> y
su latitud es <b>' + latitude + '</b></p>';
var latlng = new google.maps.LatLng(latitud, longitud);
    var myOptions = {
        zoom: 15,
        center: latlng,
        mapTypeId: google.maps.MapTypeId.ROADMAP
    };
var map = new google.maps.Map(document.getElementById
("map_canvas"), myOptions);
var marker = new google.maps.Marker({
    position: latlng,
    map: map,
    title:"Usted se encuentra aquí"
});
}
function error(pos) {
    alert('Error');
}
</script>
</head>
<body>
<h2>Html5 geolocalización</h2>
<div id="map_canvas"></div>
<div id="localisationinfo"></div>
</body>
</html>

```

Comentario

```

var latlng = new google.maps.LatLng(latitud, longitud);
var myOptions = {
    zoom: 15,
    center: latlng,
    mapTypeId: google.maps.MapTypeId.ROADMAP
};
var map = new google.maps.Map(document.getElementById("map_canvas"), myOptions);

```

El siguiente código carga Google Maps con los argumentos correspondientes a la latitud y longitud proporcionados por el API de geolocalización. Aprovechamos para definir algunas opciones, como un zoom de 15 (zoom: 15), centrado de la posición (center) y visualización en modo plano (ROADMAP).

```

var marker = new google.maps.Marker({
    position: latlng,
    map: map,
    title:"Usted se encuentra aquí"
});

```

A continuación añadimos un marcador con un título (title:"Usted se encuentra aquí"), que aparece cuando se accede a él.

Seguimiento del desplazamiento

También es posible actualizar la posición del usuario si este se mueve.

La función asíncrona `watchPosition()` devuelve una posición, pero comienza un proceso de monitorización. Si la posición del usuario se modifica, esta función devolverá inmediatamente un valor de posición. Esta función solo se aplica a los Smartphone.

Para eliminar este proceso de monitorización, usamos la función `clearWatch()`.

Para poner en marcha esta funcionalidad, es suficiente con sustituir `getCurrentPosition()`, en los puntos que hemos visto antes, por:

```
var watchId = navigator.geolocation.watchPosition(exito, error, opciones);
```

Entre estas opciones, se aconseja `enableHighAccuracy:true`, y un valor razonable para `maximumAge`.

Ejemplo

La siguiente captura de pantalla es una simulación.



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style>
div { margin-bottom: 3px; }
span { border: 1px solid silver;
       padding-left: 4px;
       padding-right: 4px; }
#boton { margin-top: 12px;
```

```
    margin-left: 60px; }
</style>
<script type="text/javascript">
function localizar(position) {
var latitud = position.coords.latitude;
var longitud = position.coords.longitude;
document.getElementById("latitud").innerHTML = latitud;
document.getElementById("longitud").innerHTML = longitud;
}
function error(error) {
switch(error.code) {
case error.UNKNOWN_ERROR:
alert("La geolocalización ha encontrado una error.");
break;
case error.PERMISSION_DENIED:
alert("El usuario no ha autorizado el acceso a su posición.");
break;
case error.POSITION_UNAVAILABLE:
alert("El usuario no se ha podido localizar.");
break;
case error.TIMEOUT:
alert("La geolocalización ha excedido el tiempo límite.");
break;
}
}
function getLocalisation(){
if(navigator.geolocation){
var watchId = navigator.geolocation.watchPosition (localizar,
error,
{enableHighAccuracy:true});
}
else {
alert("Lo sentimos, su navegador no soporta la geolocalización");
}
}
</script>
</head>
<html>
<body>
<form>
<h2>Html5 geolocalización</h2>
<div>Latitud: <span id="latitud">En espera...</span></div>
<div>Longitud: <span id="longitud">En espera...</span></div>
<input type="button" id="boton" onclick="getLocalisation();"
value="Seguir los desplazamientos">
</form>
</body>
</html>
```

Protección de la vida privada

La geolocalización puede ser indiscreta y no todos los usuarios desean compartir sus datos de geolocalización. W3C tiene reglas de uso muy estrictas.

La geolocalización debe ser voluntaria por parte del usuario. Antes de todo el procedimiento de geolocalización por parte de un sitio web, debe aparecer una ventana, cuadro de diálogo o *pop-up* en el navegador solicitando la autorización expresa del usuario.

Con la barra de autorización, el usuario final:

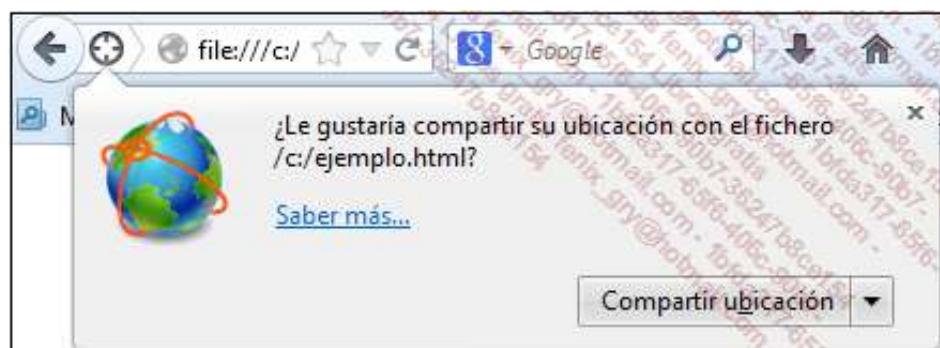
- Sabe que un sitio Web desea geolocalizarle.
- Sabe qué sitio Web quiere geolocalizarle.
- Puede elegir compartir su localización.
- Puede elegir no compartir su localización.
- Según los navegadores, puede elegir compartir momentáneamente su localización.

Por otra parte, configurando las opciones de su navegador, el usuario puede rechazar cualquier geolocalización (ver más adelante).

Google Chrome



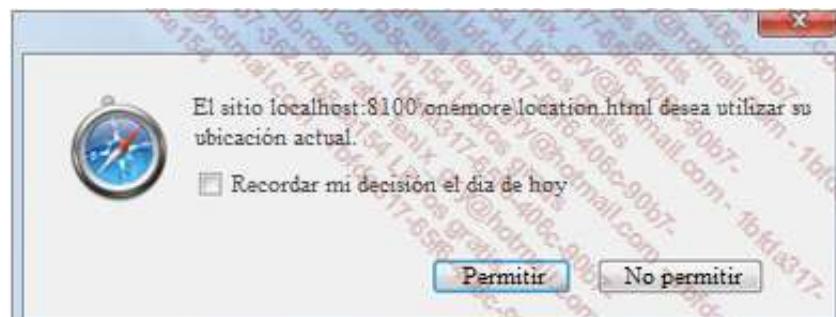
Firefox



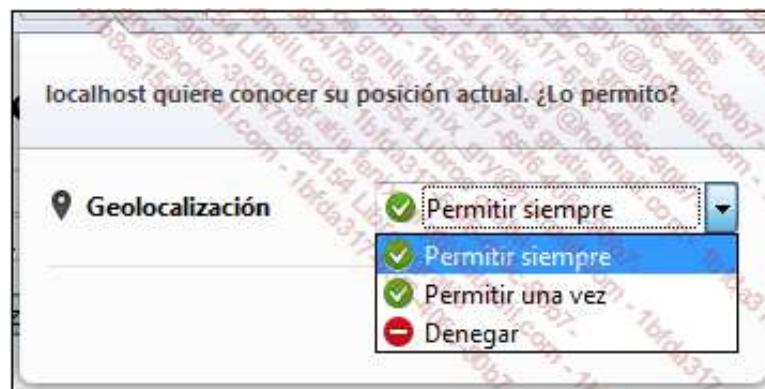
Internet Explorer



Safari



Opera

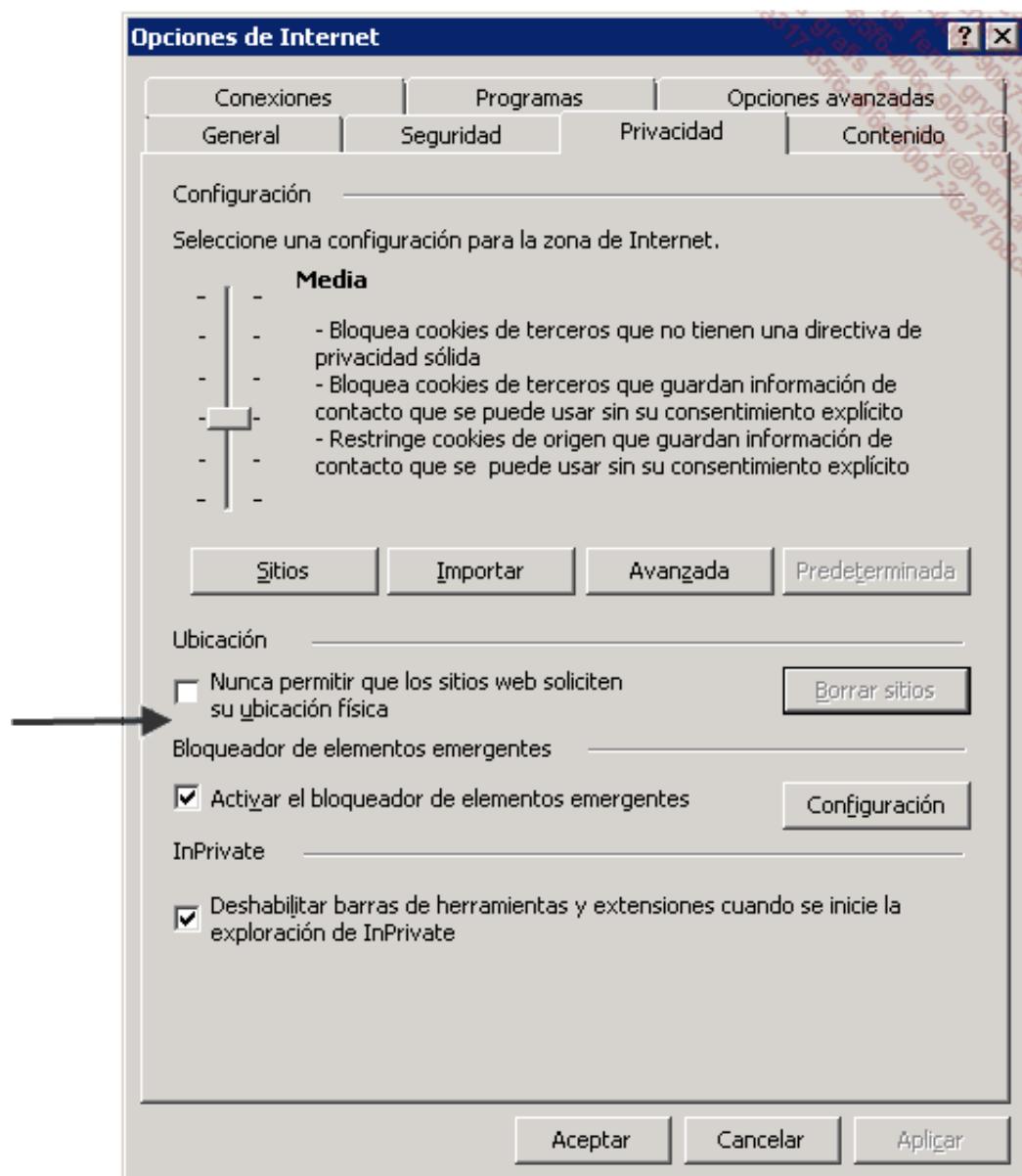


Rechazar la geolocalización

Si se advierte al usuario cada vez que un sitio Web solicita autorización para geolocalizarle de manera perceptible, rechazar cualquier geolocalización implica modificar las opciones del navegador.

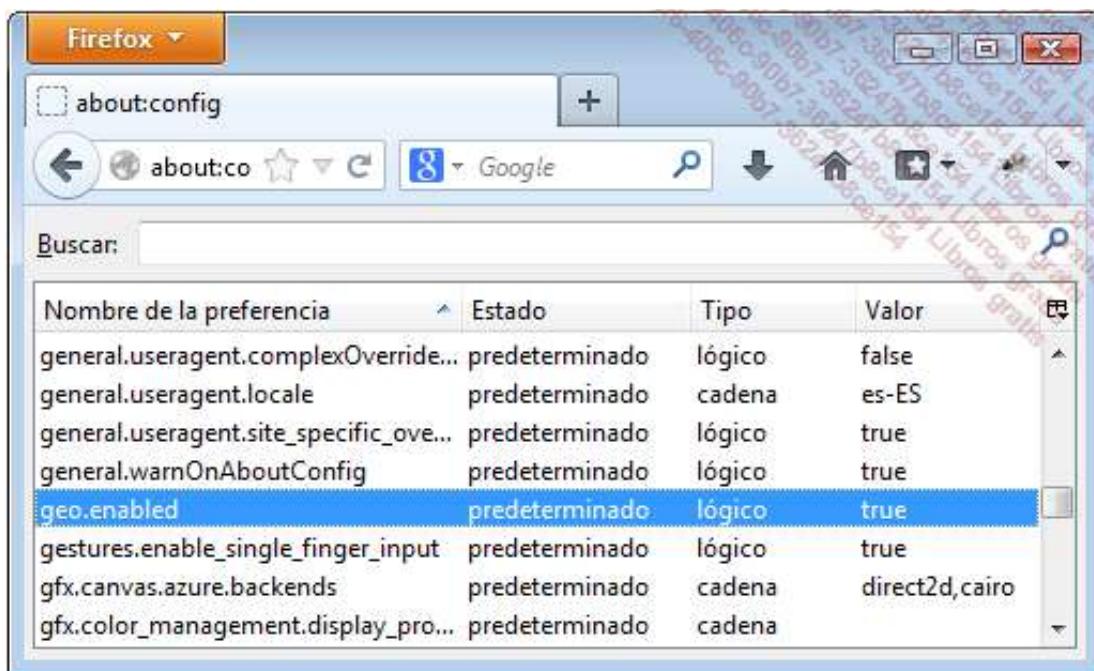
Internet Explorer 9

Herramientas - Opciones de Internet - pestaña Privacidad



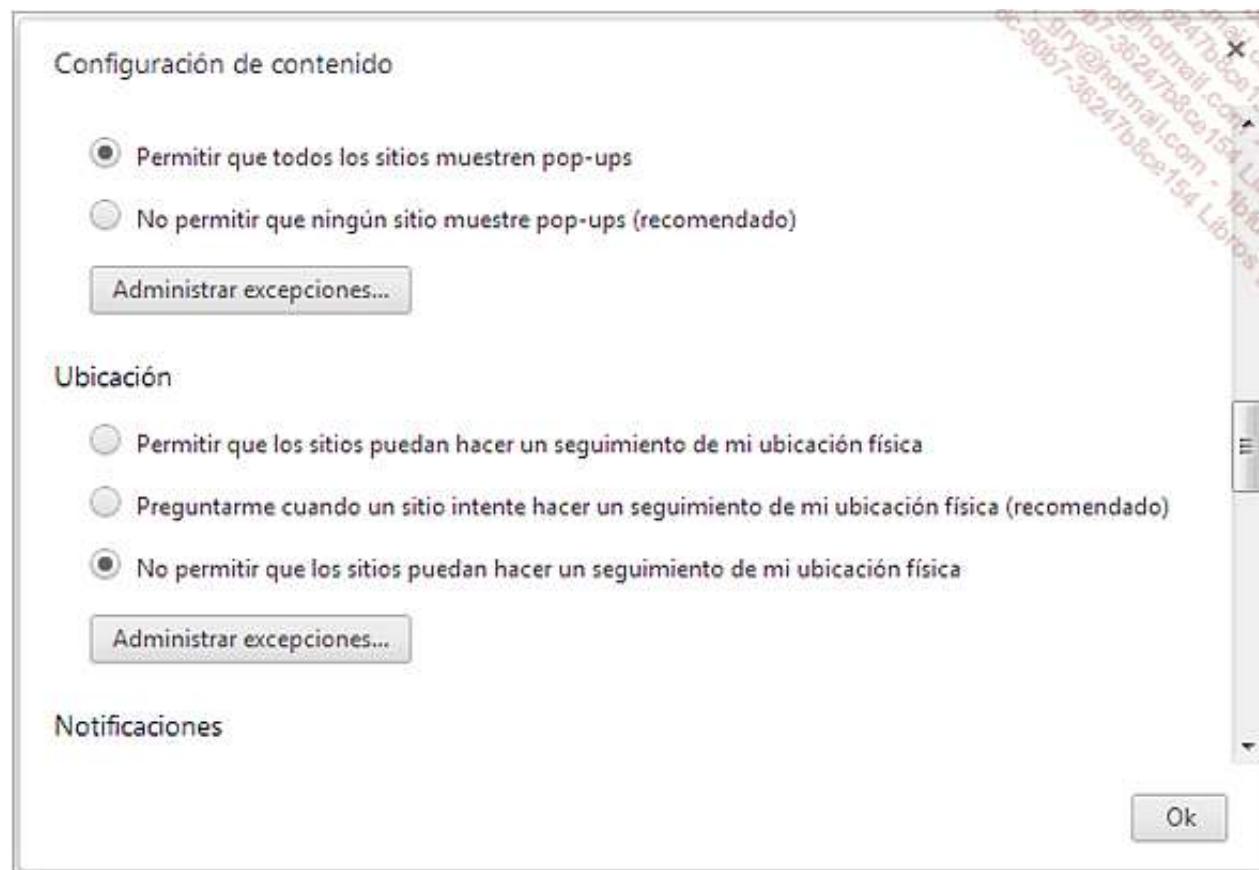
Firefox

Ejecute Firefox y a continuación escriba en la barra de dirección `about:config` y pulse la tecla [Intro]. El navegador le advierte de los inconvenientes de esta operación. Haga clic en **Tendré cuidado**. En la lista que aparece, busque el archivo `geo.enabled`. Haga doble clic sobre él y verifique que el valor `true` se convierte en `false`.



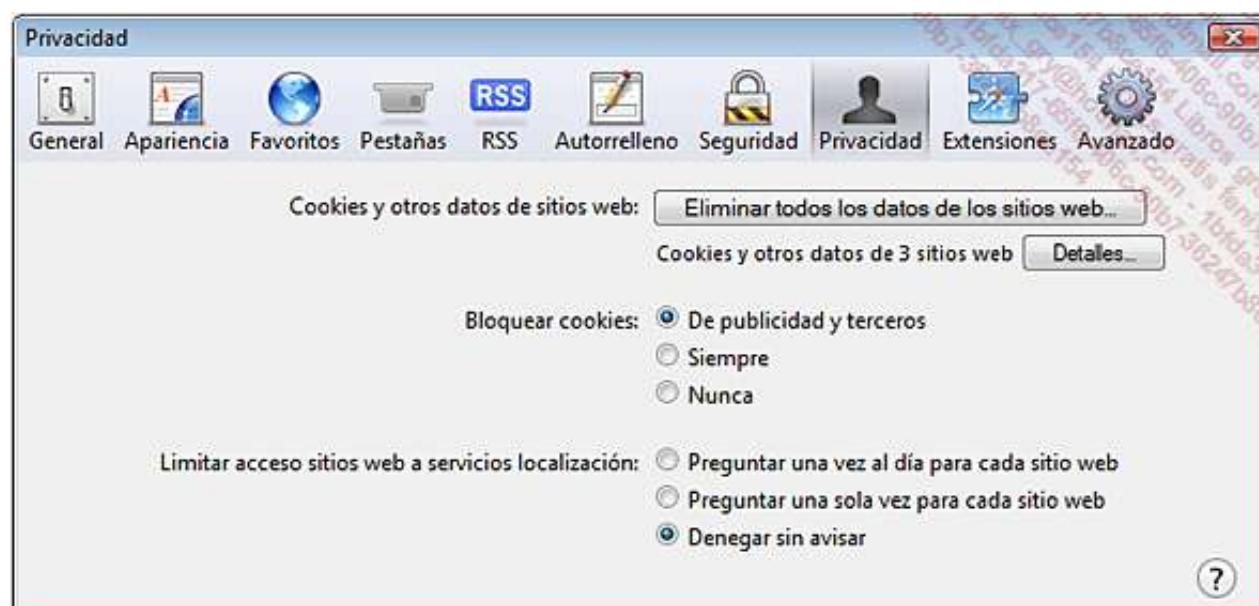
Google Chrome

Herramienta - **Configuración** - **Mostrar opciones avanzadas** - dentro de **Privacidad** - botón **Configuración del contenido** - dentro de **Ubicación** - marque **No permitir que los sitios puedan hacer un seguimiento de mi ubicación física**.



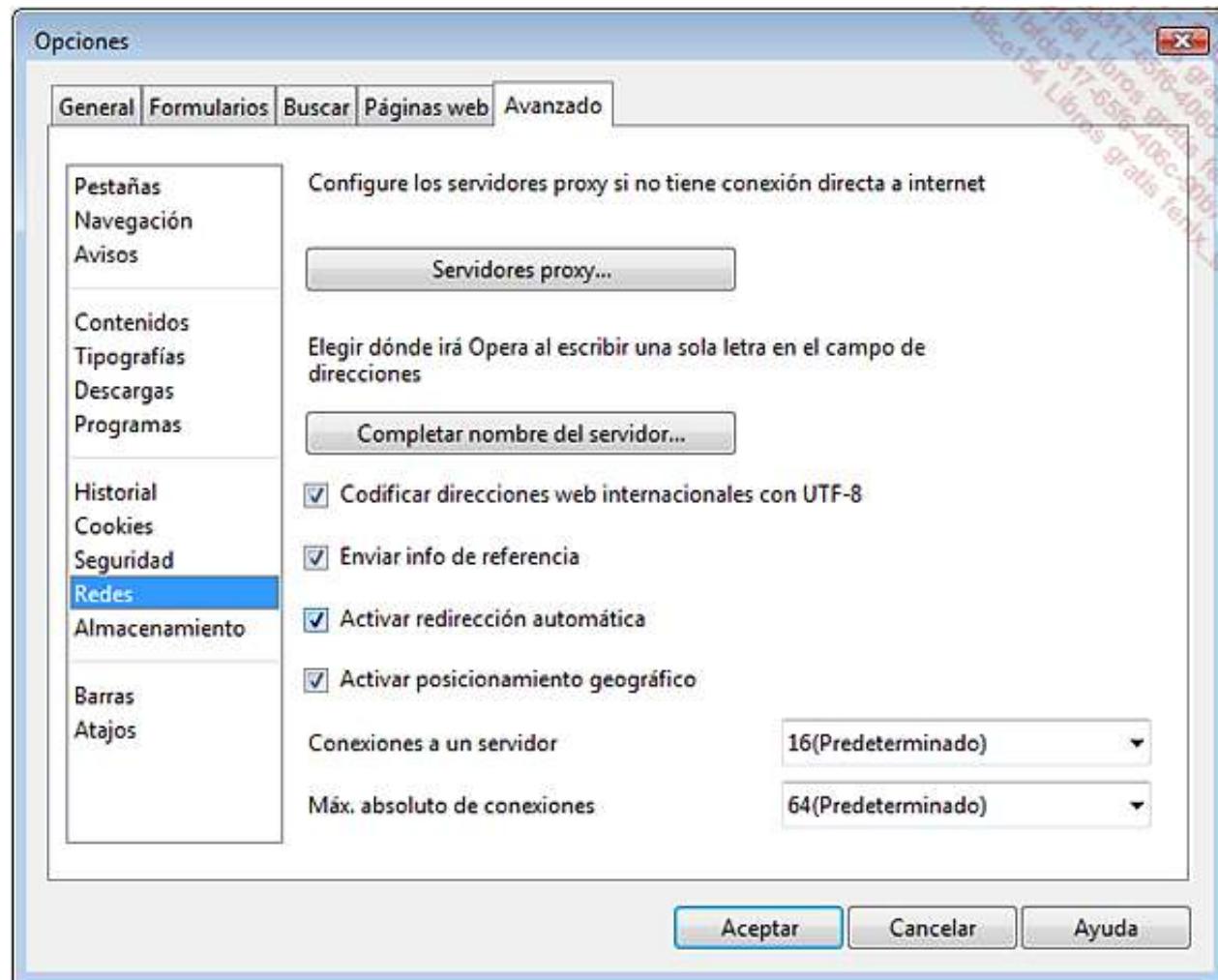
Safari 5 (para Windows)

Herramienta - **Preferencias** - pestaña **Privacidad** - sección **Limitar acceso sitios web a servicios localización** - marque la opción **Denegar sin avisar**.



Opera 11.6

Opera - **Configuración** - **Opciones** - pestaña **Avanzado** - en la lista de la izquierda **Redes**, desmarcar **Activar posicionamiento geográfico**.



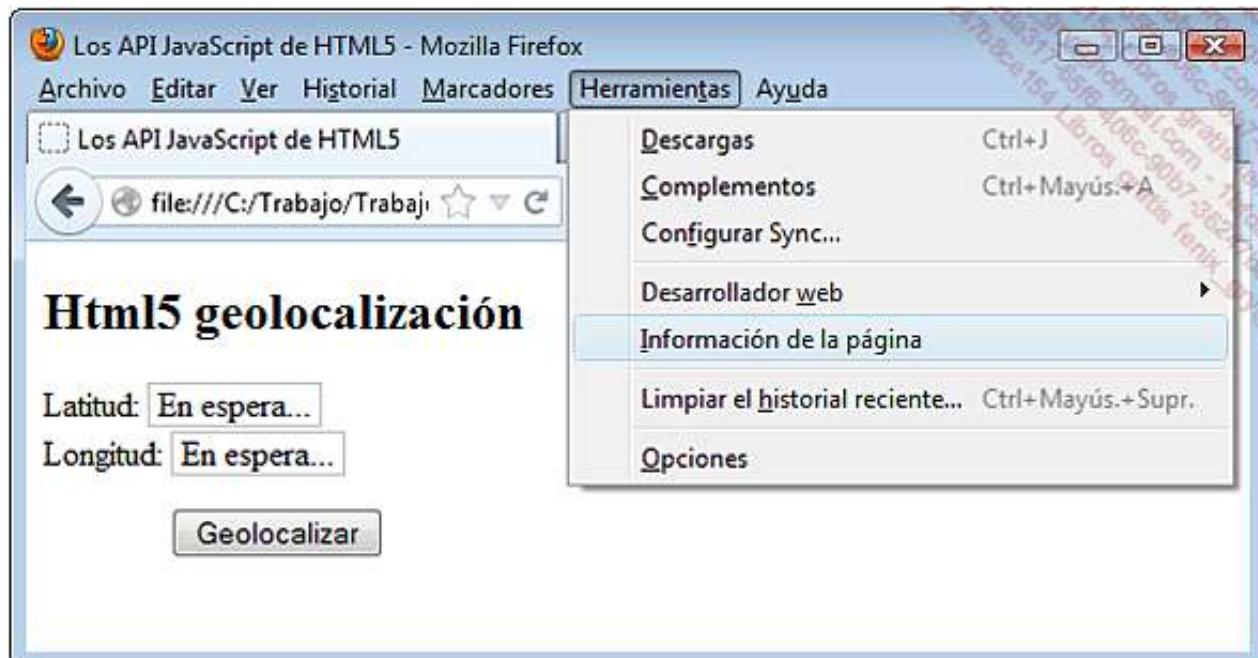
Eliminar una autorización de geolocalización

Puede suceder que hayamos dado una autorización de geolocalización y que queramos, por una razón u otra, retirar esta autorización.

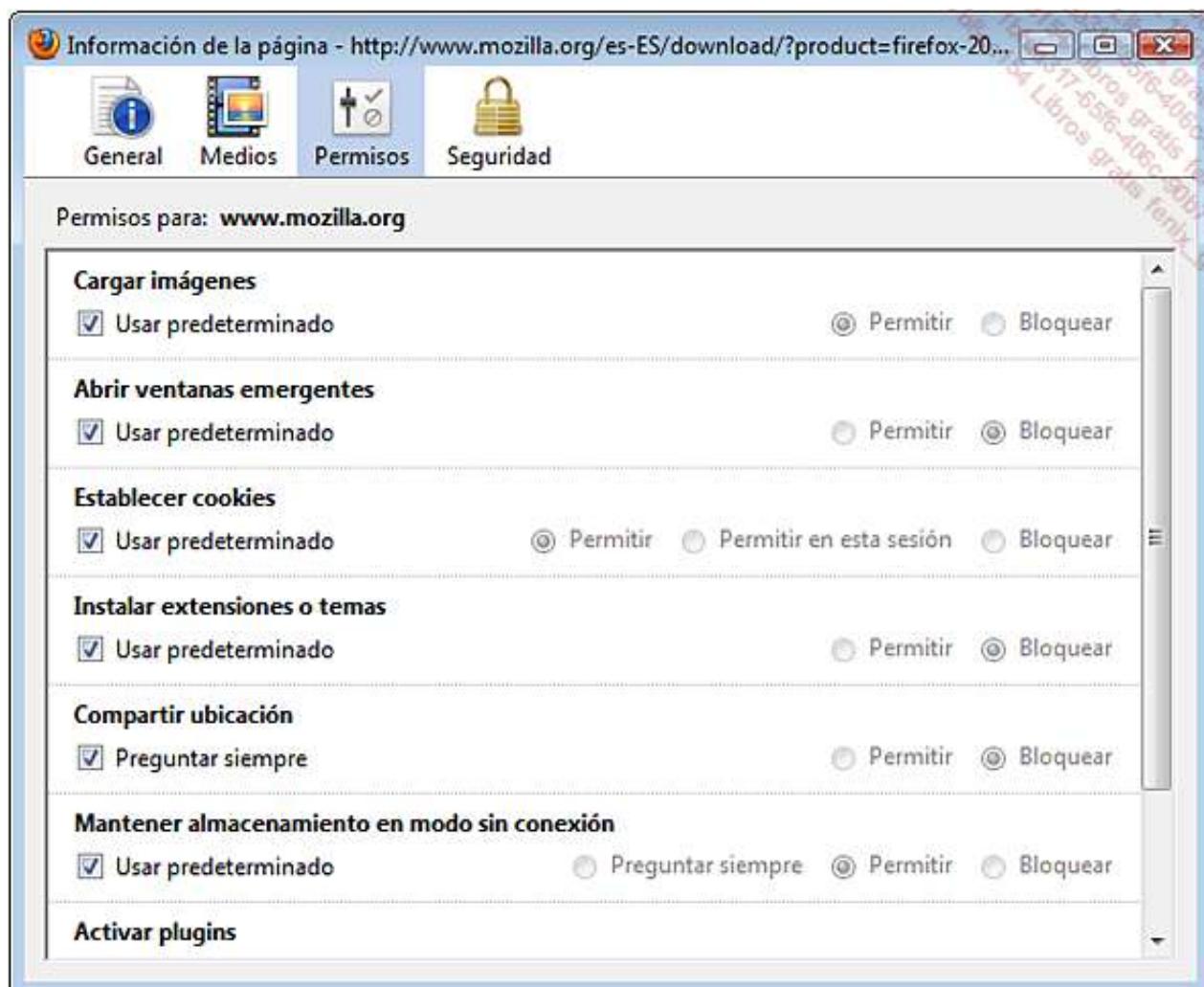
Firefox

Abra Firefox y vaya de nuevo al sitio Web donde ha dado el permiso de geolocalización.

En la barra de menú, seleccione **Herramientas - Información de la página**.



En la ventana de información de la página, seleccione la pestaña **Permisos**.

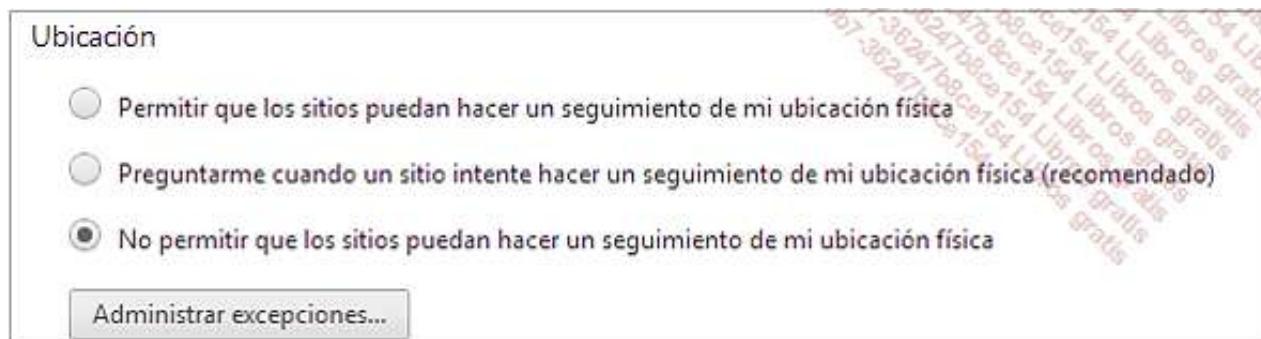


Marque el botón de radio **Bloquear** y marque la casilla de selección **Preguntar siempre**.

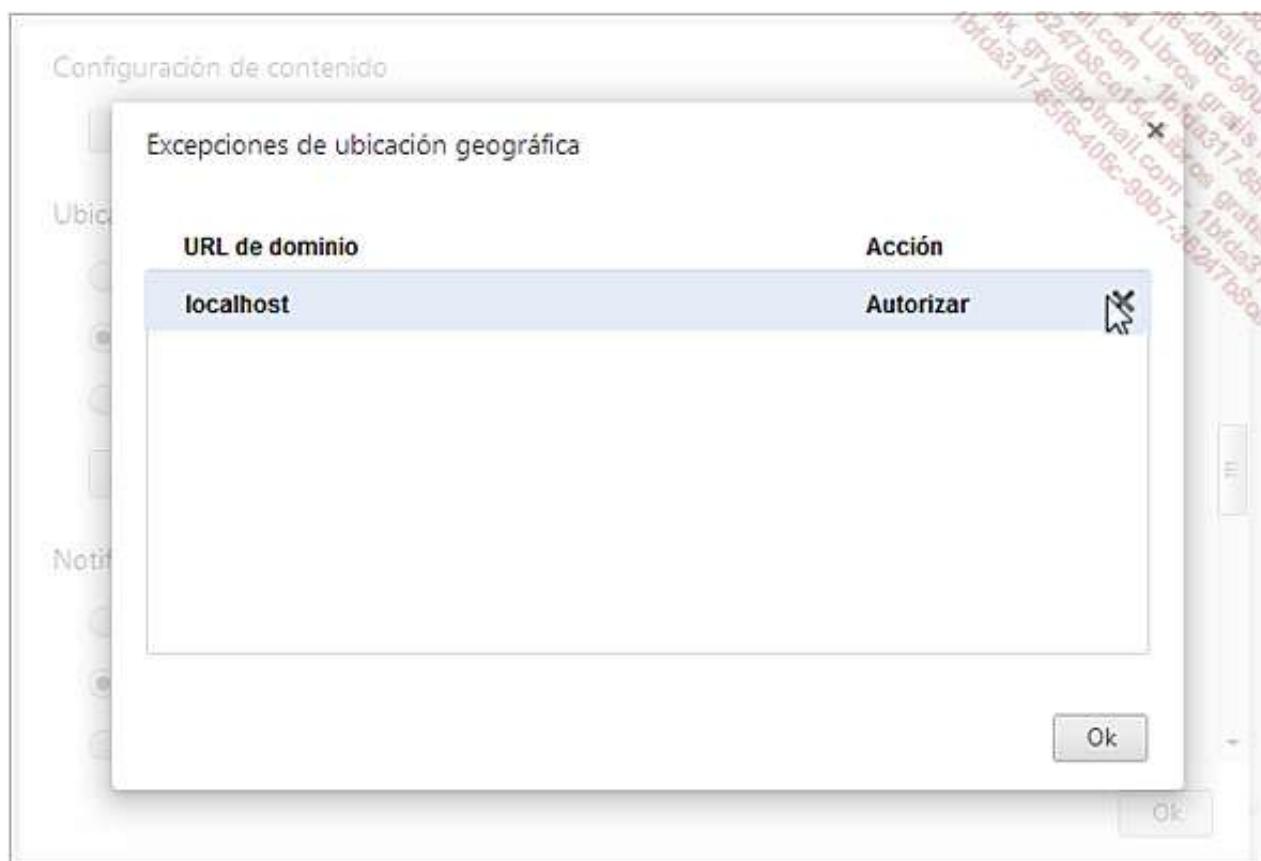


Google Chrome

Herramienta - Opciones - Mostrar opciones avanzadas - dentro de Privacidad - botón Configuración de contenido - dentro de Ubicación - botón Administrar excepciones.



Desactive la autorización pulsando en el aspa.



Internet Explorer

Internet Explorer ofrece eliminar todas las autorizaciones.

Herramientas - Opciones de Internet - pestaña Privacidad - dentro de Ubicación - botón Borrar sitios.

Opciones de Internet

Conexiones	Programas	Opciones avanzadas
General	Seguridad	Privacidad
Contenido		

Configuración

Seleccione una configuración para la zona de Internet.

Media

- Bloquea cookies de terceros que no tienen una directiva de privacidad sólida
- Bloquea cookies de terceros que guardan información de contacto que se puede usar sin su consentimiento explícito
- Restringe cookies de origen que guardan información de contacto que se puede usar sin su consentimiento explícito

Sitios **Importar** **Avanzada** **Predeterminada**

Ubicación

Nunca permitir que los sitios web soliciten su ubicación física **Borrar sitios**

Bloqueador de elementos emergentes

Activar el bloqueador de elementos emergentes **Configuración**

InPrivate

Deshabilitar barras de herramientas y extensiones cuando se inicie la exploración de InPrivate

Aceptar **Cancelar** **Aplicar**

Presentación, importancia y problemas

Entre los numerosos API propuestos por HTML5, hay un concepto particularmente innovador que destaca: es la posibilidad de almacenar en el navegador de su ordenador o su móvil una cantidad importante de información, cuando en el pasado lo más habitual era almacenarla en el lado servidor.

Este concepto de almacenamiento de información en el lado cliente se encuentra en varios API HTML5 bajo una forma u otra. Veamos algunos:

- El DOM Storage o el Web Storage; ver este capítulo.
- Una base de datos del lado cliente (Web SQL Database); ver capítulo El API Web SQL Database.
- Una base de datos del lado cliente siempre (indexed Database); ver el capítulo El API Indexed Database.
- Su asociación con la edición de contenido (contentEditable) por el usuario; ver el capítulo La edición del contenido (contentEditable).
- El uso de las aplicaciones en modo *offline*; ver el capítulo El modo desconectado (offline).

Sin ninguna duda este concepto, totalmente nuevo, va a revolucionar el diseño y desarrollo de las aplicaciones Web en los próximos años.

Sin embargo, volvamos al pasado. Para almacenar una pequeña cantidad de información, el desarrollador tenía a su disposición esta antigua (a través de la Red) herramienta que eran las cookies, que databan de Netscape 2.

Estas cookies, algunas veces muy utilizadas, solo tenía inconvenientes. En primer lugar, una capacidad reducida para contener información (alrededor de 4 KB). Por otro lado, se incluían con cada petición HTTP, lo que ralentizaba las aplicaciones Web transmitiendo los mismos datos varias veces.

A lo largo de los años, tuvieron lugar algunos intentos para mejorar esta técnica, pero que no fueron muy convincentes. Mencionamos simplemente:

- userData, disponible solo para Internet Explorer desde su versión 5.5+.
- El plugin Local Shared Object, de Adobe Flash Player.
- El plugin Google Gears para Firefox e Internet Explorer.

Con el API de almacenamiento de datos en local se abre una nueva era:

- La capacidad de información es de 5 MB (10 MB para Internet Explorer). Sabiendo que la información almacenada está en formato texto, esta zona de almacenamiento es adecuada. Observe que esta capacidad de 5 MB la ha fijado arbitrariamente el W3C.
- Esta información se almacena localmente y de esta manera es accesible desde cualquier página de la aplicación.
- Estos datos se pueden almacenar incluso si se corta la conexión con la Red (*offline*).
- Parte de esta información se puede almacenar de manera permanente en el tiempo.
- Para almacenar o explotar esta información en local, ya no es necesario hacer llamadas al servidor, lo que, en particular para los Smartphone, siempre penaliza la fluidez de la aplicación. Se ahorran muchas llamadas al servidor.

Para retomar expresiones actuales en la Red, este API Storage representa, de alguna manera, un espacio de supercookies o mejor, las cookies de Web 2.0.

Este API de almacenamiento (*Storage*), también llamado DOM Storage, tiene el nombre oficial de Web Storage (<http://dev.w3.org/html5/webstorage/>). Este nombre, sin embargo, puede dar lugar a confusión, ya que los datos no se almacenan en la Web, sino en el navegador del usuario.

Lógicamente, estos datos se almacenan por dominio. Por lo tanto, para un script que se ejecuta en un

dominio, no es posible acceder a los datos almacenados en el contexto de otro dominio.

La principal crítica de este API tiene que ver con la seguridad. De hecho, los datos se almacenan sin encriptar. De esta manera, Web Storage no es más seguro que las cookies, pero como la cantidad de información es más importante, el riesgo es más elevado. Hay varias vías abiertas para pedir que estos datos estén encriptados en el futuro, lo que se podría añadir en *undraft* futuro (borrador) de este API todavía en evolución. El modo de conexión SSL para el que cliente y servidor intercambian solo datos encriptados puede resolver en gran parte esta laguna a nivel de la seguridad. Esta es la razón por la que el protocolo https podría ser cada vez más utilizado en la Red.

Almacenamiento temporal o permanente

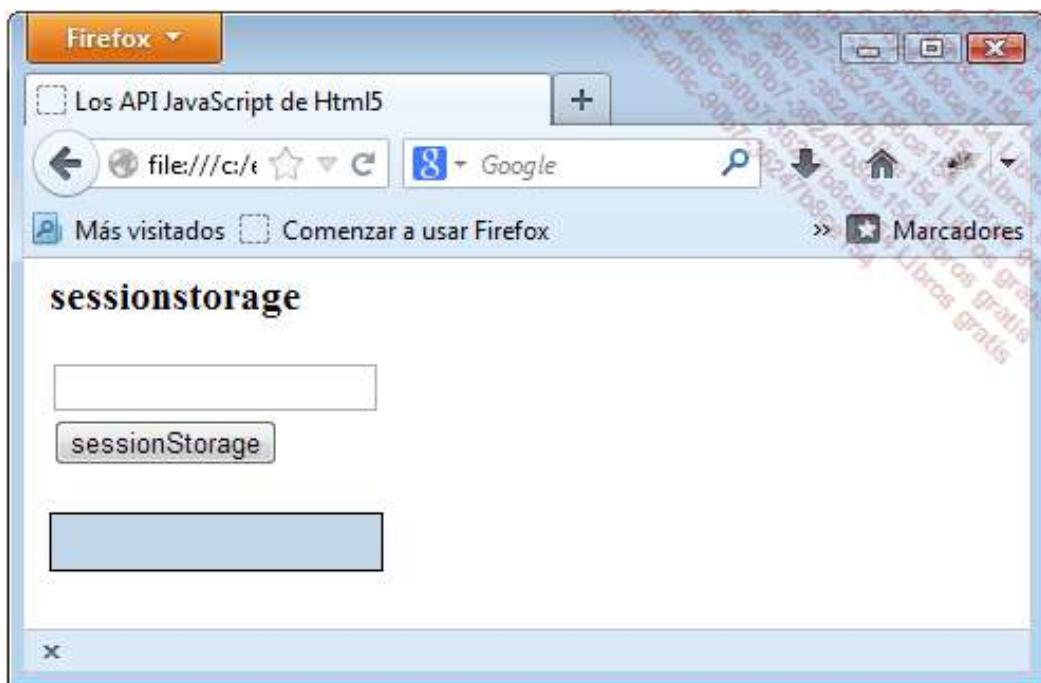
El API Storage tiene dos modos de conservar los datos: el almacenamiento temporal y el permanente. Para el devenir de este capítulo, es importante entender bien las diferencias entre estos dos modos.

1. El almacenamiento temporal

En este caso, los datos se almacenan durante la sesión actual del navegador; de ahí su nombre de **sessionStorage**. Cuando se cierra la pestaña o el navegador, estos datos almacenados ya no están disponibles.

Ilustremos esto con un ejemplo.

Dada la página inicial:



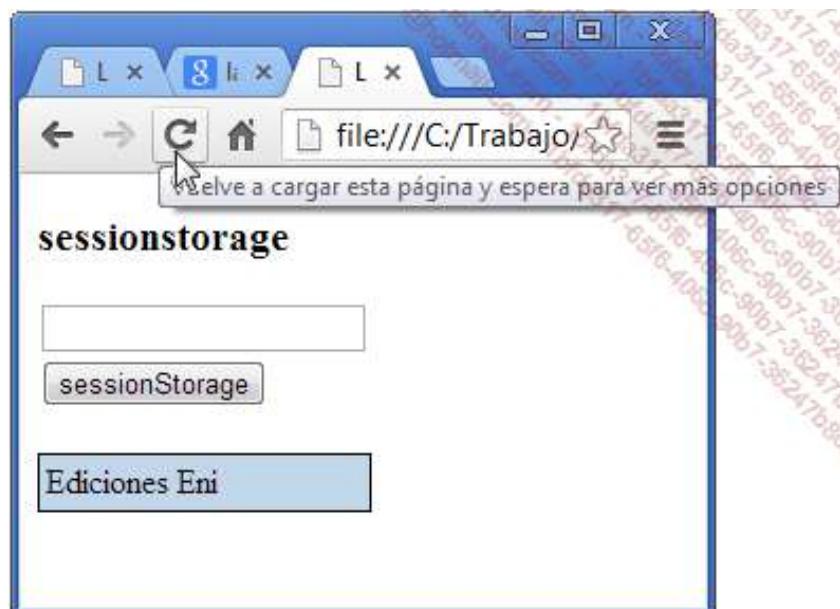
Al hacer clic en el botón, el contenido de la zona de texto se almacena de manera temporal (`sessionStorage`). Observe la capa gris.



Este se confirma con las herramientas de desarrollo de Google.

A screenshot of the Chrome DevTools interface, specifically the "Session Storage" panel under the "Elements" tab. The left sidebar lists storage types: Frames, (chap4_2_1.htm), Web SQL, IndexedDB, Local Storage, Session Storage, file:///, Cookies, and Application Cache. The "Session Storage" item is selected and highlighted in blue. The main pane shows a table with one entry: "clave" in the Key column and "Ediciones Eni" in the Value column.

El usuario puede continuar navegando por el sitio Web o actualizar la página y este dato siempre será accesible.



Por el contrario, cuando la pestaña o la ventana del navegador se cierran, los datos almacenados se eliminan y, cuando se regresa a la página, volvemos a encontrarla en su estado inicial.

The screenshot shows the Firefox developer tools open, specifically the **Elements** tab. In the left sidebar, under **Session Storage**, there is an entry for **file:///**. The main pane displays a table with two columns: **Key** and **Value**. There are four rows in the table, each containing a key and a value that appears to be a long string of characters.

A screenshot of a Firefox browser window. The address bar shows `file:///c/`. The page title is **sessionstorage**. Inside the page, there is a button labeled **sessionStorage**. Below it, a box contains the text **Ediciones Eni**.

El código de este ejemplo es:

```

<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#box { width: 150px;
    border: 1px solid black;
    margin-top: 20px;
    padding: 3px;
    background-color: rgb(195,215,235); }
</style>
<script type="text/javascript">
function storage() {
var key ="clave";
var valor = document.getElementById("in").value;
sessionStorage.setItem(key,valor);
document.getElementById("out").innerHTML =
sessionStorage.getItem("clave");
var valor = document.getElementById("in").value="";
}
function storage2() {
document.getElementById("out").innerHTML =
sessionStorage.getItem("clave");
}
</script>
</head>
<body onload="storage2()">
<h3>sessionstorage</h3>
<input type="text" id="in"><br>
<button onclick="storage()">sessionStorage</button>
<div id="box"><span id="out"></span>&ampnbsp</div>
</body>
</html>

```

El código solo se muestra como ejemplo. Su contenido se detallará más adelante en este capítulo.

2. El almacenamiento permanente

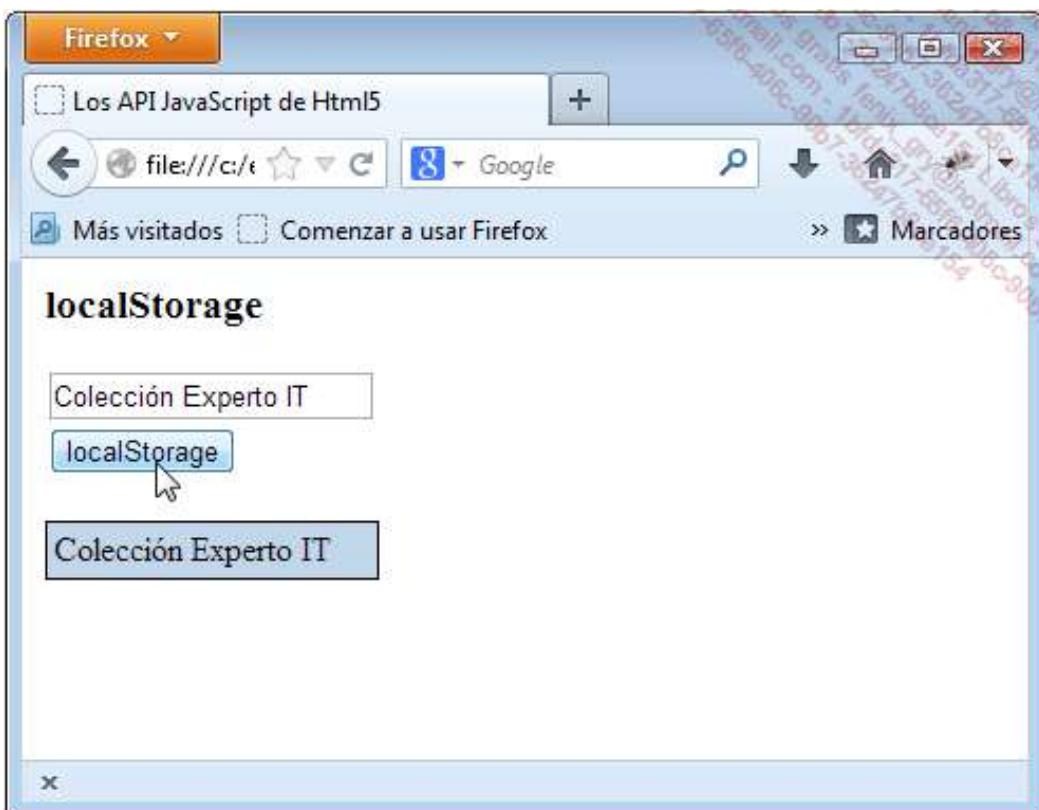
Aquí los datos se almacenan en local de manera indefinida, de ahí su nombre de **localStorage**. Cuando se cierra la pestaña o el navegador, estos datos guardados siguen estando disponibles.

Veamos un ejemplo.

Consideremos la misma página inicial, pero adaptada para el localStorage:



Al hacer clic en el botón, el contenido de la zona de texto se almacena permanentemente (localStorage).



Screenshot of the Firefox Developer Tools Network tab showing Local Storage data.

	Key	Value
clave		Colección Experto IT

El usuario puede cerrar el navegador. Una hora, un día, un mes o un año después, si el usuario regresa a esta página, los datos almacenados seguirán estando presentes y disponibles.



Esta vez el código es:

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#box { width: 150px;
        border: 1px solid black;
        margin-top: 20px;
        padding: 3px;
        background-color: rgb(195,215,235); }
</style>
<script type="text/javascript">
function storage() {
var key ="clave";
var valor = document.getElementById("in").value;
localStorage.setItem(key,valor);
document.getElementById("out").innerHTML =

```

```

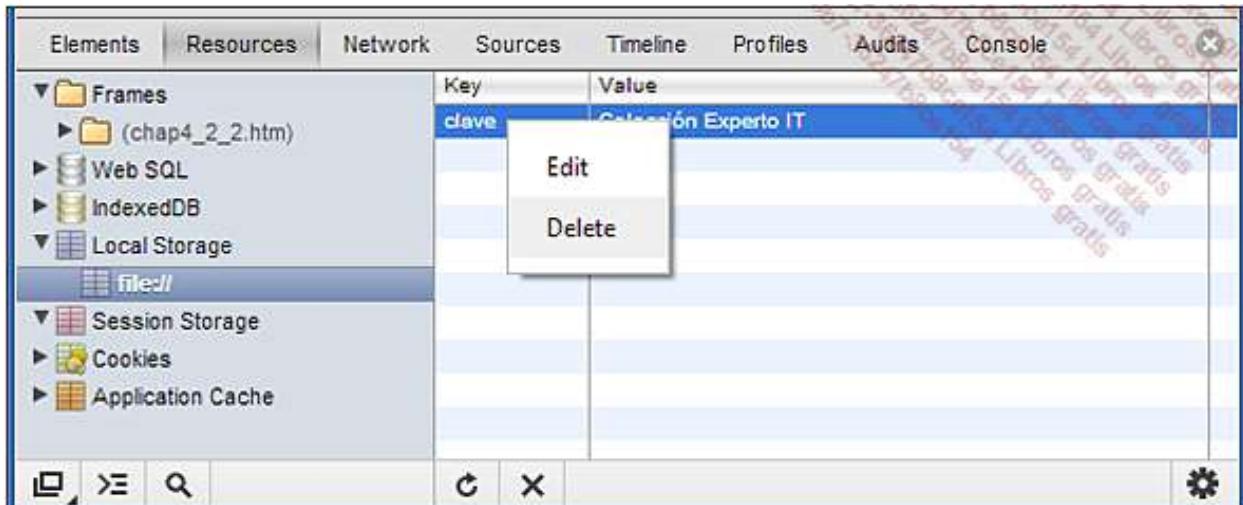
localStorage.getItem("clave");
var valor = document.getElementById("in").value="";
}
function storage2() {
document.getElementById("out").innerHTML =
localStorage.getItem("clave");
}
</script>
</head>
<body onload="storage2()">
<h3 >localStorage</h3>
<input type="text" id="in"><br>
<button onclick="storage()">localStorage</button>
<div id="box"><span id="out"></span>&ampnbsp</div>
</body>
</html>

```

Este código solo se muestra como ejemplo. Su contenido se verá en detalle más adelante en este capítulo.

Observe simplemente que, respecto al código de almacenamiento temporal, se han remplazado todos los casos de sessionStorage por localStorage.

- Mientras esté en periodo de aprendizaje o desarrollo, la memoria localStorage será muy solicitada. Para no encontrar los datos de un ejemplo o intento anterior, algunas veces será necesario vaciar los datos de esta zona de almacenamiento permanente (selección de los elementos - botón derecho del ratón - **Eliminar**).



Disponibilidad del API

El API Web Storage está muy implantado en los navegadores actuales. Por ejemplo, para los ordenadores de escritorio:

- Internet Explorer 8+.
- Firefox 3.6+.
- Chrome 4.0+ para el localStorage. Chrome 5.0+ para sessionStorage.
- Safari 4.0+.
- Opera 10.5+.

Y para los Smartphone y otras tabletas:

- iPhone 2.0+.
- Android 2.0+.
- Opera Móvil 10.0.

Web Storage es uno de los API HTML5 que ya se puede utilizar con total seguridad, porque los navegadores actuales lo soportan completamente.

Sin embargo, para las aplicaciones Web con un requerimiento de compatibilidad máxima, es prudente probar la disponibilidad. También veremos que el usuario puede elegir desactivar esta función de almacenamiento.

El código JavaScript verificará la existencia de las propiedades sessionStorage o localStorage del objeto global window.

También es necesario saber que, en todos los navegadores recientes, el almacenamiento temporal y el permanente se implementan al mismo tiempo. Por lo tanto, para probar la función Storage, es indiferente probar la propiedad sessionStorage o localStorage.

Método 1

Si el navegador no soporta el API HTML5 Web Storage, la propiedad localStorage estará indefinida. Como consecuencia de un bug desafortunado de una versión antigua de Firefox (Firefox 3.6), esta prueba genera una excepción si las cookies están desactivadas. De esta manera, la prueba de disponibilidad tiene que utilizar una instrucción try ... catch.

```
function supports_local_storage() {  
try {  
    return 'localStorage' in window && window['localStorage'] !== null;  
} catch(e){  
    return false;  
}  
}
```

De manera pragmática, nos podemos conformar con:

```
if ("localStorage" in window && window["localStorage"] != null) {  
// LocalStorage está soportada  
}
```

Ejemplo

```
<!DOCTYPE html>  
<html lang="es">  
<head>  
<title>Los API JavaScript de HTML5</title>  
<meta charset=utf-8>
```

```

<script type="text/javascript">
function supportlocalStorage() {
if ("localStorage" in window && window["localStorage"] != null) {
alert("La función Html5 Storage está soportada")
}
else {
alert('Su navegador no soporta Storage');
}
}
window.onload = function() {
supportlocalStorage();
}
</script>
</head>
<body>
<h1>Html5 Storage</h1>
</body>
</html>

```



Este script funciona en local.

Método 2

El segundo método consiste en probar directamente `window.localStorage` o `window.sessionStorage` en una prueba condicionada (`if`).

Fíjese en que este script se debe ejecutar en un servidor local.

Ejemplo

```

<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<script type="text/javascript">
function check() {
if (window.localStorage) {

```

```

        alert("La función Html5 Storage está soportada");
    }
    else {
        alert('Su navegador no soporta localStorage');
    }
    if (window.sessionStorage) {
        alert('La función Html5 sessionStorage está soportada ');
    } else {
        alert('Su navegador no soporta sessionStorage');
    }
}
</script>
</head>
<body onload="check()">
<h1>Html5 Storage</h1>
</body>
</html>

```

El resultado en un emulador de Opera Mini es:

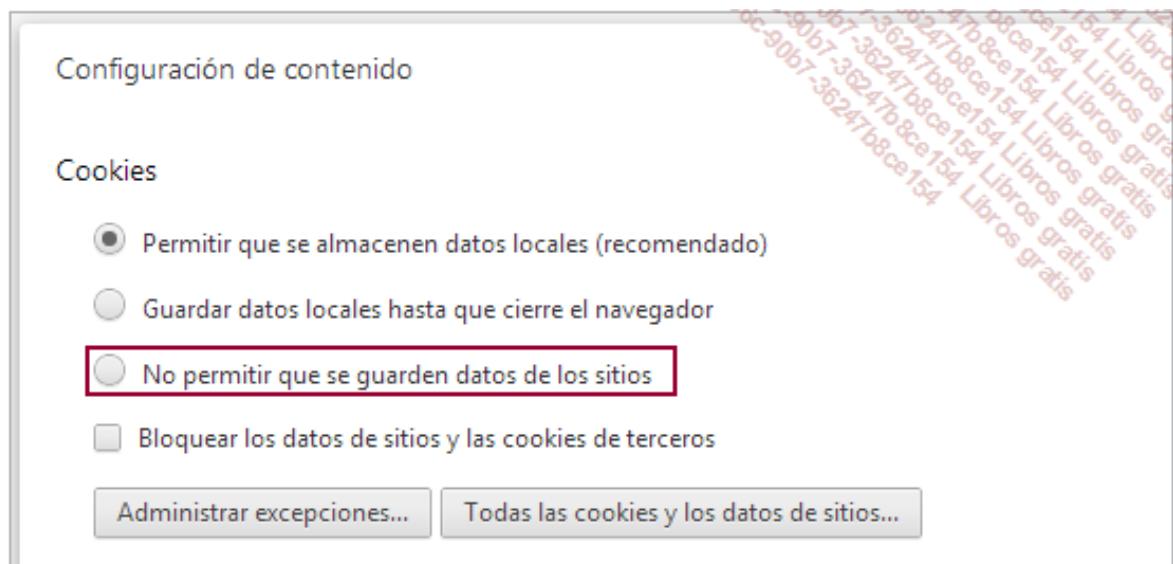


Comentario

Por opción personal o para evitar operaciones malintencionadas, el usuario puede desactivar la función Storage.

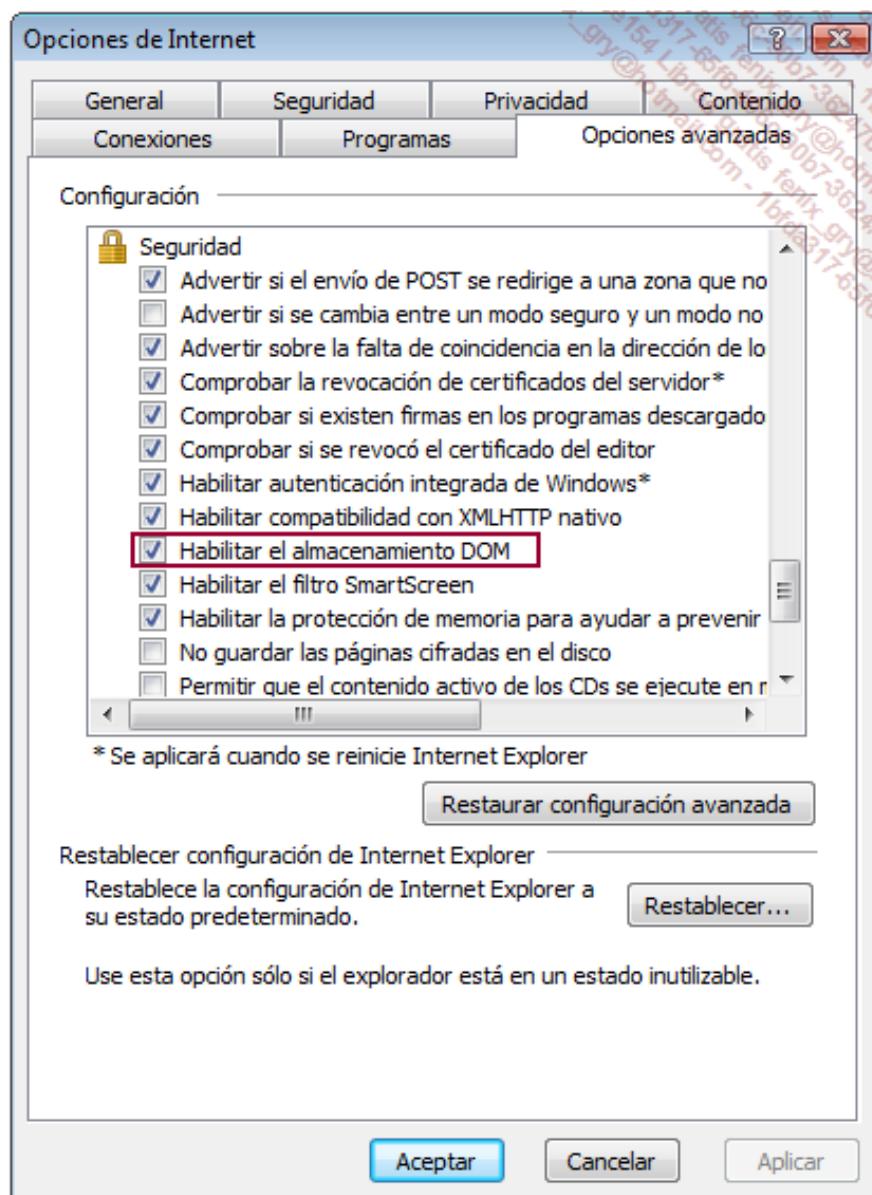
En Google Chrome

En Herramientas - **Configuración** - **Mostrar opciones avanzadas** - dentro de **Privacidad** - botón **Configuración de contenido - Cookies**.



En Internet Explorer 9

Herramientas - **Opciones de Internet** - pestaña **Opciones avanzadas**:



En Firefox

Para modificar las preferencias, escriba `about:config` en la barra de dirección. En la larga lista que se presenta, localice `dom.storage.enabled`. El valor debe estar a `true` para la activación. El valor es `false` cuando está desactivado.

Nombre de la preferencia	Estado	Tipo	Valor
<code>dom.mozNetworkStats.enabled</code>	predetermi...	lógico	<code>false</code>
<code>dom.mozPermissionSettings.enabled</code>	predetermi...	lógico	<code>false</code>
<code>dom.mozSettings.enabled</code>	predetermi...	lógico	<code>false</code>
<code>dom.network.enabled</code>	predetermi...	lógico	<code>true</code>
<code>dom.network.metered</code>	predetermi...	lógico	<code>false</code>
<code>dom.placeholder.show_on_focus</code>	predetermi...	lógico	<code>true</code>
<code>dom.popup_allowed_events</code>	predetermi...	cadena	<code>change</code>
<code>dom.popup_maximum</code>	predetermi...	entero	<code>20</code>
<code>dom.send_after_paint_to_content</code>	predetermi...	lógico	<code>false</code>
<code>dom.server-events.default-reconnection-time</code>	predetermi...	entero	<code>5000</code>
<code>dom.server-events.enabled</code>	predetermi...	lógico	<code>true</code>
<code>dom.sms.enabled</code>	predetermi...	lógico	<code>false</code>
<code>dom.sms.strict7BitEncoding</code>	predetermi...	lógico	<code>false</code>
<code>dom.storage.default_quota</code>	predetermi...	entero	<code>5120</code>
<code>dom.storage.enabled</code>	predetermi...	lógico	<code>true</code>

El almacenamiento permanente (localStorage)

1. Añadir un valor

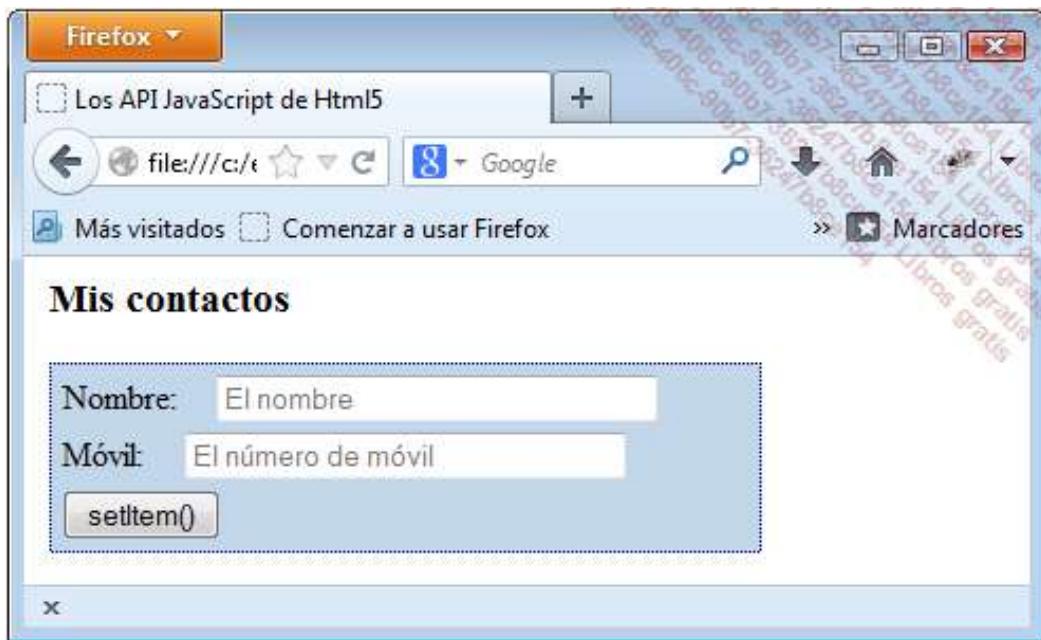
La adición de un dato permanente se hace con la propiedad `setItem()` del objeto `localStorage`. Estos datos forman parejas (clave, valor). Observe que estos pares son cadenas de caracteres y pueden estar vacías. La sintaxis es la siguiente:

```
localStorage.setItem(clave, valor);
```

De esta manera, los números se almacenan en formato de cadena de caracteres. Para utilizarlos como números, será necesario usar las funcionalidades `parseInt`, `parseFloat` o `Number` del JavaScript clásico.

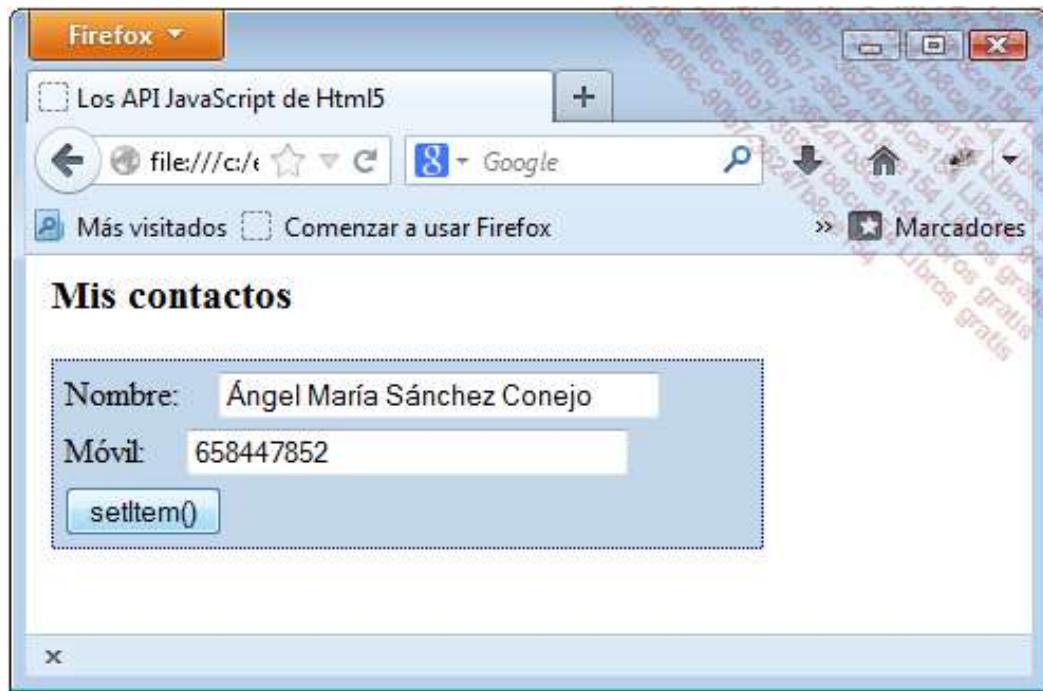
Ejemplo

Vamos a construir un directorio telefónico que estará disponible de manera permanente en su ordenador o Smartphone.



Para seguir los ejemplos relativos al almacenamiento permanente, vamos a codificar los siguientes valores:

- Ángel María Sánchez Conejo 658447852
- María González-Aller Zavala 658778420
- José Antonio Sánchez Fresneda 687118950
- Mateo Sánchez González-Aller 685159851



Para los incrédulos, vamos a hacer las comprobaciones con la herramienta de desarrollo de Google Chrome (acceso directo [Ctrl][Shift] I). Los datos codificados están presentes en **LocalStorage - Local Files**.

A screenshot of the Google Chrome DevTools Storage panel. The left sidebar shows a tree view with "Frames", "Web SQL", "IndexedDB", and "LocalStorage". Under "LocalStorage", there is an entry for "file://ejemplo.html". The main pane is a table with columns "Key" and "Value". It lists four items: "José Antonio Sánchez Fresneda" with value "687118950", "María González-Aller Zavala" with value "658778420", "Mateo Sánchez González-Aller" with value "685159851", and "Ángel María Sánchez Conejo" with value "658447852". The "LocalStorage" entry for "file://ejemplo.html" is highlighted.

El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#box { border: 1px dotted navy;
        padding-left: 5px;
        background-color: rgb(195,215,235);
        width: 330px; }
p { margin-top: 5px;
    margin-bottom: 5px; }
</style>
<script type="text/javascript">
function dosetItem() {
var name = document.getElementById("nombre").value;
var data = document.getElementById("movil").value;
```

```

localStorage.setItem(name, data);
document.getElementById("nombre").value="";
document.getElementById("movil").value="";
}
</script>
</head>
<body>
<header>
<h3>Mis contactos </h3>
</header>
<section>
<form>
<div id="box">
<p>
<label for="nombre">Nombre:</label>
<input id="nombre" name="nombre" size="30" placeholder="El nombre"
style="margin-left: 15px;">
</p>
<p>
<label for="movil">Móvil:</label>
<input id="movil" name="movil" size="30" placeholder="El número
de móvil">
</p>
<p>
<input type="button" value="setItem()" onclick="dosetItem()">
</p>
</div>
</form>
</section>
</body>
</html>

```

Comentario

```

var name = document.getElementById("nombre").value;
var data = document.getElementById("movil").value;

```

Las variables name y data contienen respectivamente los valores de las zonas de texto dedicadas al nombre y al número móvil.

```
localStorage.setItem(name, data);
```

Estos datos se añaden (setItem()) a la zona de almacenamiento permanente (localStorage). La variable name sirve de clave y la variable data de valor.

 Los diferentes pares (clave, valor) forman una matriz de tipo Array. Así, en caso de que sea necesario, podemos añadir una pareja (clave, valor) en formato:localStorage["clave"] = "valor";

2. Recuperar un valor almacenado

La propiedad getItem() permite localizar un valor existente. Su sintaxis es:

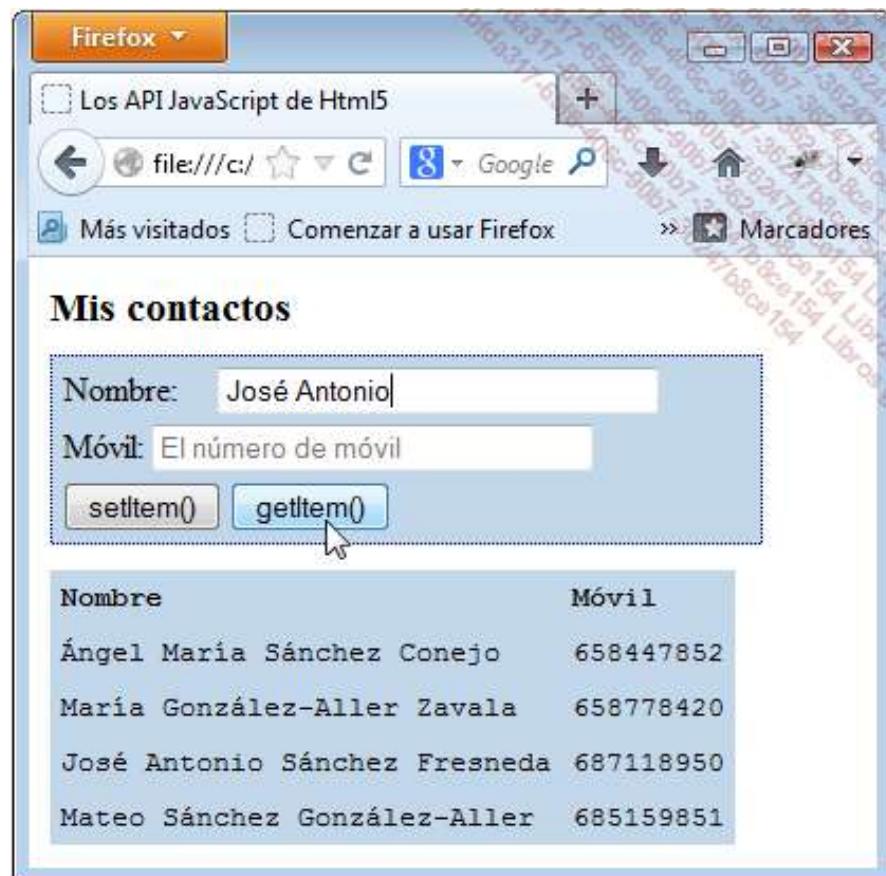
```
localStorage.getItem(clave);
```

Ejemplo

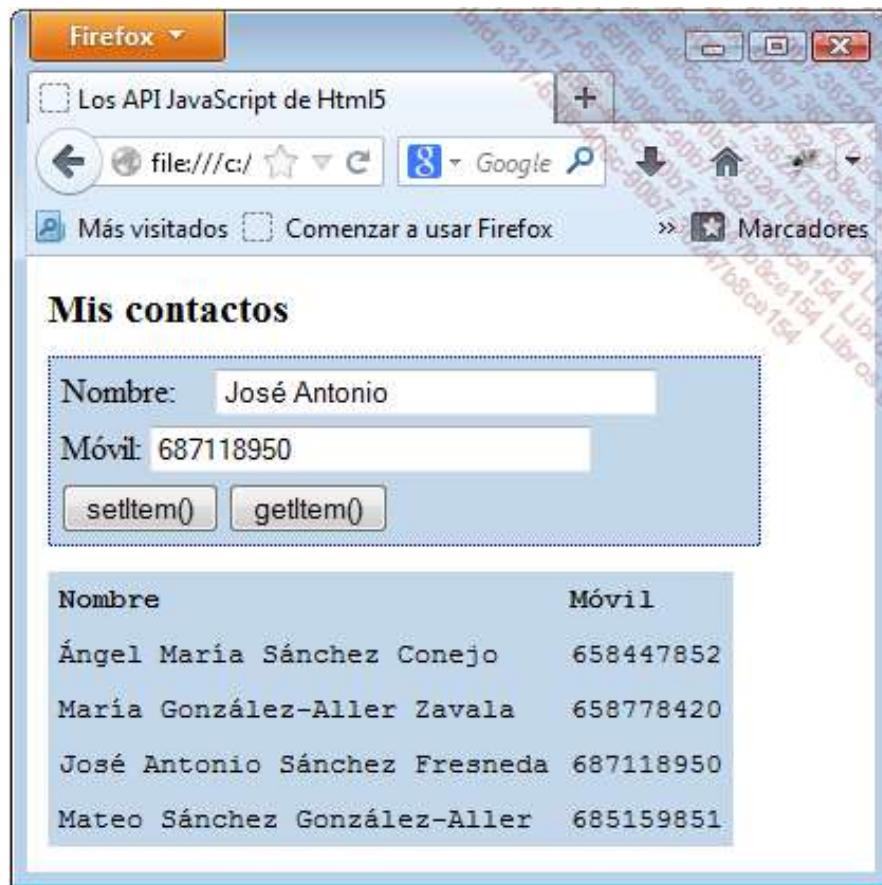
Retomemos nuestro directorio del ejemplo anterior. Habíamos añadido una función para ver los datos almacenados y mejorar su visualización.



Para localizar el número móvil de José Antonio, es suficiente con codificar esta clave en la línea de texto relativa al nombre.



Y aparece el número de móvil (es decir, el valor).



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#box { border: 1px dotted navy;
       padding-left: 5px;
       background-color: rgb(195,215,235);
       width: 330px;}
p { margin-top: 5px;
    margin-bottom: 5px;}
td, th { font-family: monospace;
          padding: 5px;
          background-color: rgb(195,215,235);}
table { margin-top: 12px;
        border-spacing: 0px;}
</style>
<script type="text/javascript">
function dosetItem() {
var name = document.getElementById("nombre").value;
var data = document.getElementById("movil").value;
localStorage.setItem(name, data);
document.getElementById("nombre").value="";
document.getElementById("movil").value="";
visualizar();
}
function dogetItem() {
var name = document.getElementById("nombre").value;
document.getElementById("movil").value =
localStorage.getItem(name);
visualizar();
}
```

```

function visualizar() {
var key = "";
var pares = "<tr><td><b>Nombre</b></td><td><b>Móvil</b></td>
</tr>\n";
var i=0;
for (i=0; i<=localStorage.length-1; i++) {
key = localStorage.key(i);
pares +=
"<tr><td>" +key+ "</td>\n<td>" +localStorage.getItem(key) +"</td>
</tr>\n";
}
if (pares == "<tr><td><b>Nombre</b></td><td><b>Móvil</b></td>
</tr>\n") {
pares += "<tr><td><i>Vacio</i></td>\n<td><i>vacio</i></td>
</tr>\n";
}
document.getElementById('pares').innerHTML = pares;
}
</script>
</head>
<body onload="visualizar()">
<header>
<h3>Mis contactos</h3>
</header>
<section>
<form>
<div id="box">
<p>
<label for="nombre">Nombre:</label>
<input id="nombre" name="nombre" size="30" placeholder="El nombre"
style="margin-left: 15px;">
</p>
<p>
<label for="movil">Móvil:</label>
<input id="movil" name="movil" size="30" placeholder="El número
de móvil">
</p>
<p>
<input type="button" value="setItem()" onclick="dosetItem()">
<input type="button" value="getItem()" onclick="dogetItem()">
</p>
</div>
</form>
<table id=pares></table>
</section>
</body>
</html>

```

Comentario

```

function dogetItem() {
var name = document.getElementById("nombre").value;
document.getElementById("movil").value = localStorage.getItem(name);
visualizar();
}

```

La variable name recupera el valor de la zona de texto identificada por nombre (document.getElementById("nombre").value).

La zona de texto móvil (document.getElementById("movil").value) toma como valor el devuelto por localStorage.getItem, al que le hemos asignado la variable name como clave.

La función visualizar() muestra el nuevo estado del directorio.

Esta función de visualización (visualizar()) está escrita en su mayor parte en JavaScript clásico.

Sin embargo, algunos elementos merecen comentarse.

```
for (i=0; i<=localStorage.length-1; i++) {
```

El atributo `length` devuelve el número de pares clave/valor del objeto `localStorage`.

```
key = localStorage.key(i);
```

El método `key(n)` del objeto `localStorage` devuelve la clave correspondiente al índice `n`. Si no se encuentra ningún valor, el valor es `NULL`.

3. Modificar un valor almacenado

No existe ninguna propiedad específica para modificar un valor almacenado. Es suficiente con llamar a la clave del valor que se va a modificar (`getItem()`), codificar el nuevo valor y almacenar la pareja (`setItem()`). Esto elimina el valor antiguo.

Observe que, en fase de desarrollo, es posible modificar directamente la clave y el valor en Web Inspector de Google.

Key	Value
José Antonio Sánchez Fresneda	687118950
Maria González-Aller Zavala	658778420
Mateo Sánchez González	685159851
Ángel María Sánchez Co...	658447852

4. Eliminar un valor almacenado

Para eliminar un valor, el desarrollador dispone de la propiedad `removeItem()`.

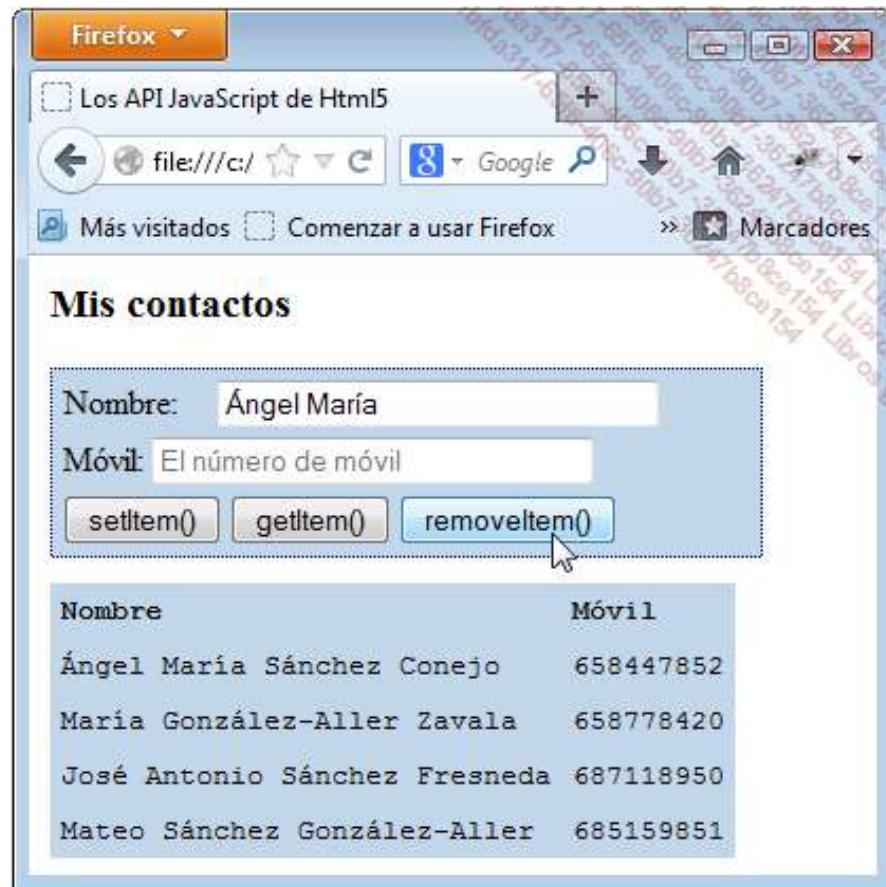
```
localStorage.removeItem(clave);
```

De esta manera, se eliminarán la clave que se da como argumento y su valor asociado. Si la clave no existe, no se elimina nada.

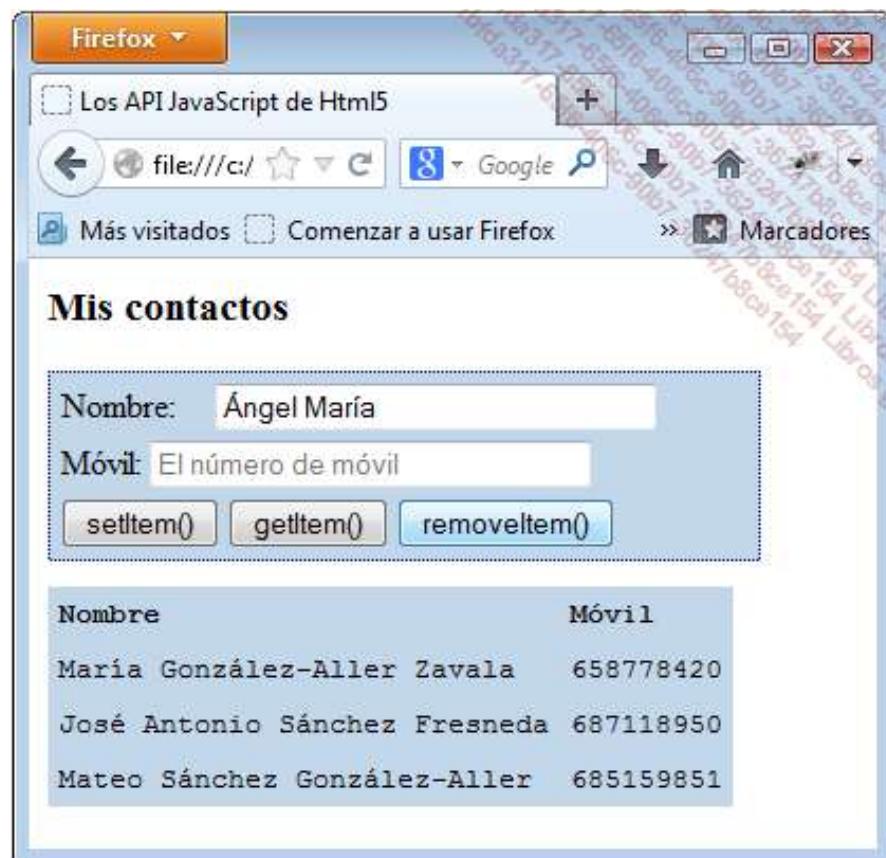
Ejemplo

Vamos a eliminar de nuestro directorio el ítem Ángel María y el número de móvil que le corresponde.

Después de codificar Ángel María, es suficiente con pulsar el botón `removeItem()` para eliminarlo de nuestra lista.



El resultado es el siguiente:



[El código](#)

```
<!DOCTYPE html>
```

```
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#box { border: 1px dotted navy;
        padding-left: 5px;
        background-color: rgb(195,215,235);
        width: 330px;}
p { margin-top: 5px;
    margin-bottom: 5px;}
td, th { font-family: monospace;
        padding: 5px;
        background-color: rgb(195,215,235);}
table { margin-top: 12px;
        border-spacing: 0px;}
</style>
<script type="text/javascript">
function dosetItem() {
var name = document.getElementById("nombre").value;
var data = document.getElementById("movil").value;
localStorage.setItem(name, data);
document.getElementById("nombre").value="";
document.getElementById("movil").value="";
visualizar();
}
function dogetItem() {
var name = document.getElementById("nombre").value;
localStorage.getItem(name);
visualizar();
}
function doremoveItem() {
var name = document.getElementById("nombre").value;
localStorage.removeItem(name);
visualizar();
}
function visualizar() {
var key = "";
var pares = "<tr><td><b>Nombre</b></td><td><b>Móvil</b></td>
</tr>\n";
var i=0;
for (i=0; i<=localStorage.length-1; i++) {
key = localStorage.key(i);
pares += "<tr><td>" + key + "</td>\n<td>" + localStorage.getItem(key) + "</td>
</tr>\n";
}
if (pares == "<tr><td><b>Nombre</b></td><td><b>Móvil</b></td>
</tr>\n") {
pares += "<tr><td><i>Vacio</i></td>\n<td><i>vacio</i></td>
</tr>\n";
}
document.getElementById('pares').innerHTML = pares;
}
</script>
</head>
<body onload="visualizar()">
<header>
<h3>Mis contactos</h3>
</header>
<section>
<form>
<div id="box">
<p>
<label for="nombre">Nombre:</label>
<input id="nombre" name="nombre" size="30" placeholder="El nombre">
</p>
<label for="movil">Número:</label>
<input id="movil" name="movil" type="tel" size="30" placeholder="El número">
</div>
<table border="1">
<thead>
<tr>
<th>Nombre</th>
<th>Número</th>

```

```

style="margin-left: 15px;">>
</p>
<p>
<label for="movil">Móvil:</label>
<input id="movil" name="movil" size="30" placeholder="El número
de móvil">
</p>
<p>
<input type="button" value="setItem()" onclick="dosetItem()">
<input type="button" value="getItem()" onclick="dogetItem()">
<input type=button value="removeItem()">
onclick="doremoveItem()">
</p>
</div>
</form>
<table id=pares></table>
</section>
</body>
</html>

```

Comentario

```

function doremoveItem() {
var name = document.getElementById("nombre").value;
localStorage.removeItem(name);
visualizar();
}

```

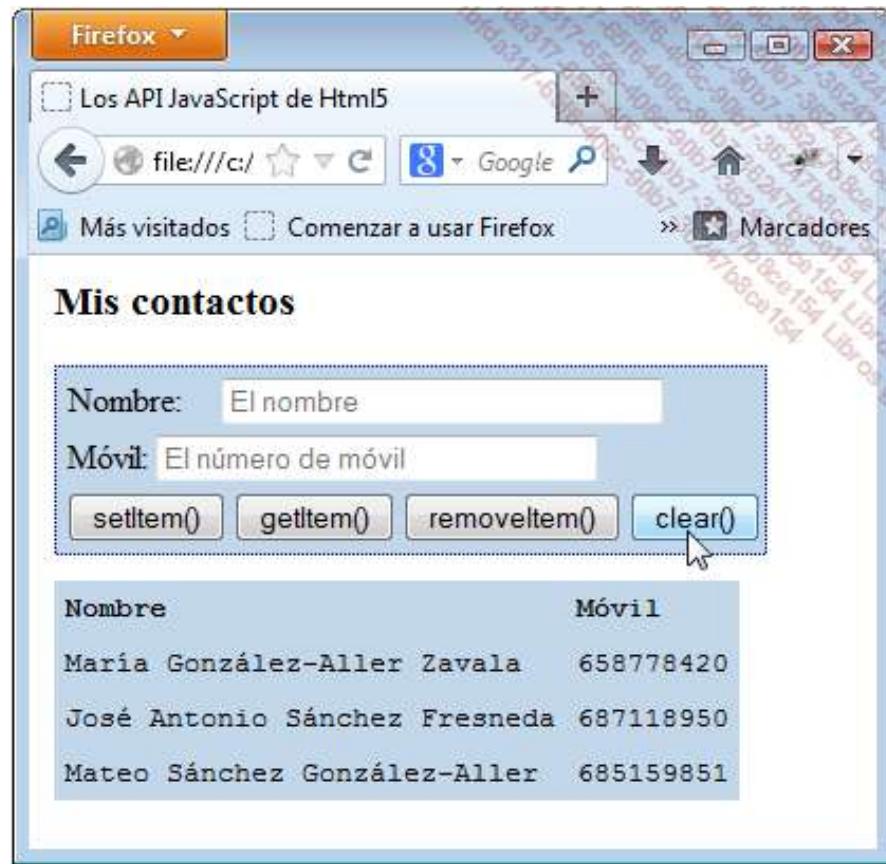
La función `doremoveItem()` comienza recuperando, en la variable `name`, el valor codificado en la línea de texto identificada por `nombre`. Esta variable `name` es un argumento de la propiedad `localStorage.removeItem(name)`. La función `visualizar()` muestra el nuevo estado del directorio.

5. Eliminar todos los valores almacenados

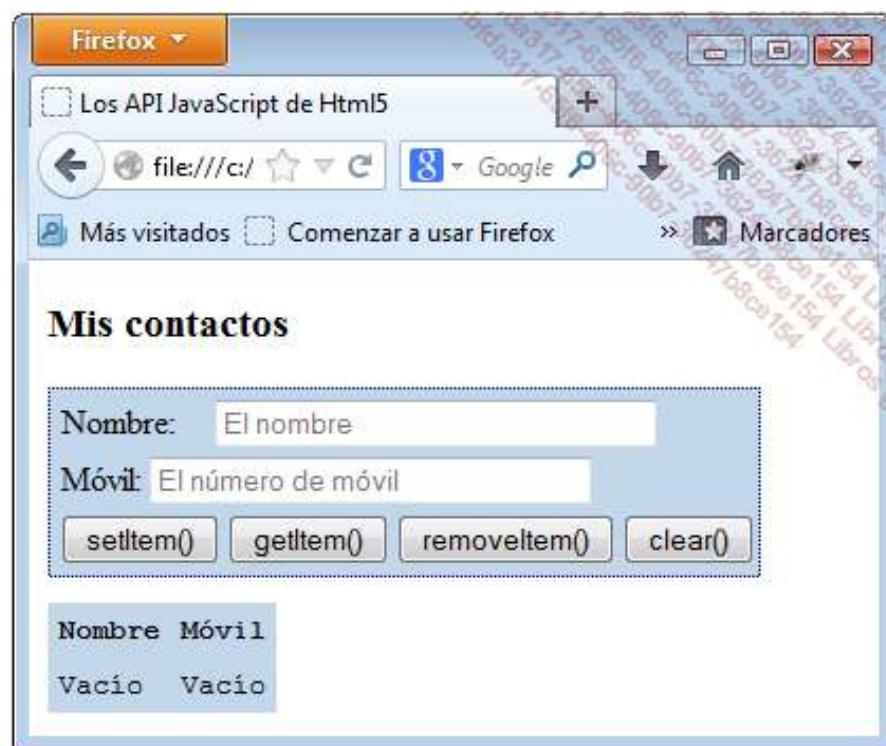
Para eliminar todos los pares clave/valor, el desarrollador dispone de la propiedad `clear()`.

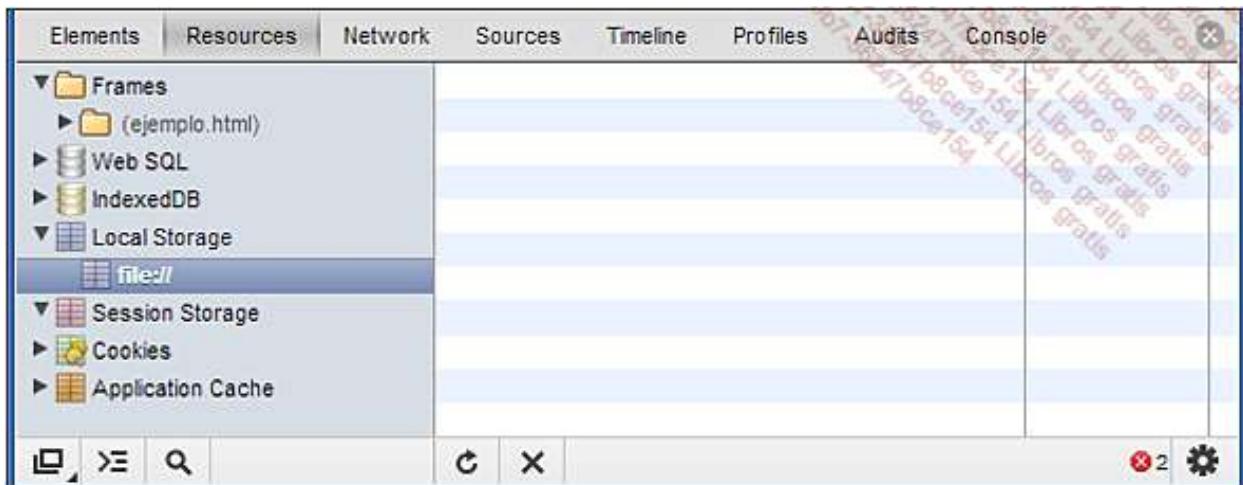
```
localStorage.clear();
```

Ejemplo



El resultado es el siguiente:





El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#box { border: 1px dotted navy;
       padding-left: 5px;
       background-color: rgb(195,215,235);
       width: 330px; }
p { margin-top: 5px;
    margin-bottom: 5px; }
td, th { font-family: monospace;
          padding: 5px;
          background-color: rgb(195,215,235); }
table { margin-top: 12px;
        border-spacing: 0px; }
</style>
<script type="text/javascript">
function dosetItem() {
var name = document.getElementById("nombre").value;
var data = document.getElementById("movil").value;
localStorage.setItem(name, data);
document.getElementById("nombre").value="";
document.getElementById("movil").value="";
visualizar();
}
function dogetItem() {
var name = document.getElementById("nombre").value;
localStorage.getItem(name);
visualizar();
}
function doremoveItem() {
var name = document.getElementById("nombre").value;
localStorage.removeItem(name);
visualizar();
}
function doClear() {
localStorage.clear();
visualizar();
}
function visualizar() {
var key = "";
var pares = "<tr><td><b>Nombre</b></td><td><b>Móvil</b></td>
</tr>\n";
for (var i = 0; i < localStorage.length; i++) {
key = localStorage.key(i);
pares += "<tr><td>" + key + "</td><td>" + localStorage.getItem(key) + "</td>
</tr>\n";
}
document.getElementById("box").innerHTML = pares;
}
</script>
```

```

var i=0;
for (i=0; i<localStorage.length-1; i++) {
key = localStorage.key(i);
pares += "<tr><td>" + key + "</td>\n<td>" + localStorage.getItem(key) + "</td>
</tr>\n";
}
if (pares == "<tr><td><b>Nombre</b></td><td><b>Móvil</b></td>
</tr>\n") {
pares += "<tr><td><i>Vacío</i></td>\n<td><i>Vacio</i></td>
</tr>\n";
}
document.getElementById('pares').innerHTML = pares;
}
</script>
</head>
<body onload="visualizar()">
<header>
<h3>Mis contactos</h3>
</header>
<section>
<form>
<div id="box">
<p>
<label for="nombre">Nombre:</label>
<input id="nombre" name="nombre" size="30" placeholder="El nombre"
style="margin-left: 15px;">
</p>
<p>
<label for="movil">Móvil:</label>
<input id="movil" name="movil" size="30" placeholder="El número
de móvil">
</p>
<p>
<input type="button" value="setItem()" onclick="dosetItem()">
<input type="button" value="getItem()" onclick="dogetitem()">
<input type=button value="removeItem()"
onclick="doremoveItem()">
<input type=button value="clear()" onclick="doClear()">
</p>
</div>
</form>
<table id=pares></table>
</section>
</body>
</html>

```

6. Aplicación final

Imagine una lista de compras.



Codificamos algunos productos:



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
ul { list-style-type: none; }
li { margin-left: -20px; }
#box { width: 200px;
```

```

border: 1px solid black;
-webkit-border-radius: 5px;
-moz-border-radius: 5px;
border-radius: 5px;
-moz-box-shadow: 1px 1px 12px #555;
-webkit-box-shadow: 1px 1px 12px #555;
box-shadow: 1px 1px 12px #555; }

.nuevo { margin-top: 20px; }

</style>
<script type="text/javascript">
function añadir(){
var all = 'todo'
var almacen = document.getElementById("producto").value
if (localStorage.todo){
almacen= localStorage.getItem('todo') + "<li>" + almacen + "</li>";
}
else{
almacen = "<li>" + almacen + "</li>"
}
localStorage.setItem(todo, almacen);
}

function editar(){
if ( localStorage.getItem('todo') ) {
document.getElementById('edit').innerHTML =
localStorage.getItem('todo');
}
}

function initial(){
localStorage.clear();
}

</script>
</head>
<body onload="editar()">
<header>
<h1> Mi lista de compras</h1>
</header>
<section>
<form>
<p>
<input type="text" id="producto" size="28"
placeholder="Para comprar">
<button onclick="añadir()">Añadir</button>
</p>
<div id="box">
<ul id="edit">
</ul>
</div>
<button class="nuevo" onclick="initial()">Nueva lista
</button>
</form>
</section>
</body>
</html>

```

Comentario

La siguiente captura de pantalla de Web Inspector ilustra la técnica adoptada por este script.

Key	Value
Todo	 Algo de fruta Judías verdes Pechuga de pollo Agua mineral

El script solo crea una clave (todo) cuyo valor es una serie de etiquetas ... correspondiente a la lista de productos.

```
function añadir(){
var all = 'todo'
```

Creación de la variable todo, que será la clave.

```
var almacen = document.getElementById("producto").value
```

La variable almacen corresponde al producto codificado en la zona de texto.

```
if (localStorage.todo) {
almacen= localStorage.getItem('todo') + "<li>" + almacen + "</li>";
}
else{
almacen = "<li>" + almacen + "</li>";
}
localStorage.setItem(todo, almacen);
}
```

Si la clave todo ya existe en localStorage (if(localStorage.todo), la variable almacen se convierte en el valor anterior (localStorage.getItem('todo')), al que añadimos un ítem de lista que contiene el producto añadido por la variable almacen ("" + almacen + ""). Si la clave todo todavía no existe, añadimos simplemente las etiquetas de ítem de lista ("" + almacen + "") a la variable almacen. La clave todo y el valor almacen se añaden entonces a la memoria local (localStorage.setItem(todo, almacen)).

```
function editar(){
if (localStorage.getItem('todo')) {
document.getElementById('edit').innerHTML =
localStorage.getItem('todo');
}
}
```

La visualización de la lista (editar()), con la condición de que la clave todo tenga un valor (if (localStorage.getItem('todo'))), añade (innerHTML) entre las etiquetas el valor asociado a todo (localStorage.getItem('todo')).

```
function initial(){
localStorage.clear();
}
```

La función initial() asociada al botón Nueva lista vacía la memoria (localStorage.clear()).

El almacenamiento temporal (`sessionStorage`)

A nivel de código JavaScript, el diseño de los scripts es prácticamente igual. Es suficiente con sustituir `localStorage` por `sessionStorage`. Las propiedades son totalmente idénticas.

1. Añadir un valor

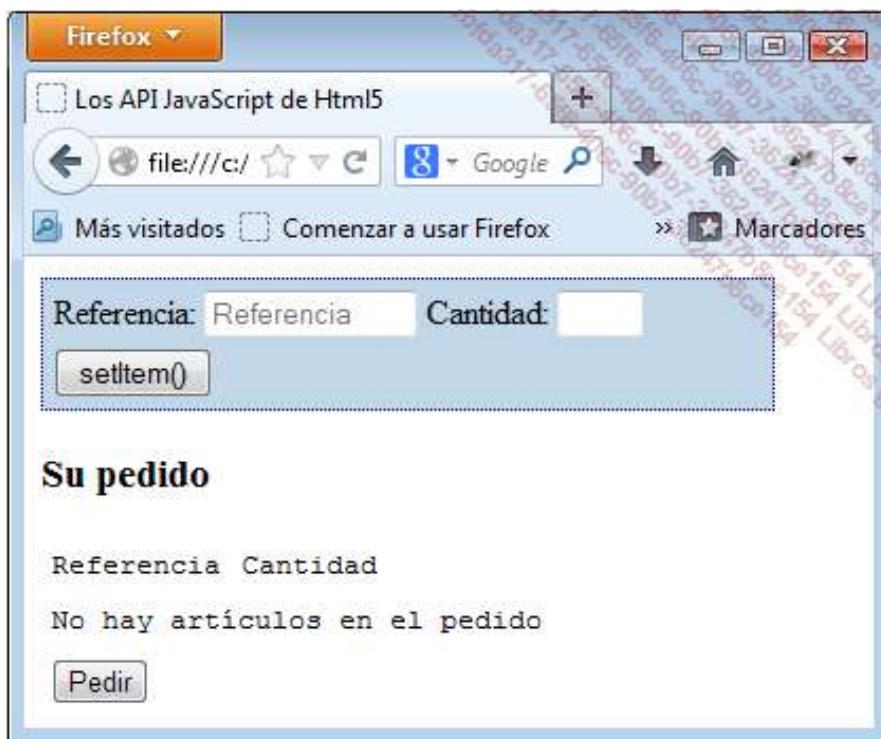
La adición de un dato temporal se hace con la propiedad `setItem()` del objeto `sessionStorage`. Estos datos tienen la forma de pares (clave, valor). Observe que estos pares son cadenas de caracteres y pueden estar vacías.

```
sessionStorage.setItem(clave, valor);
```

Ejemplo

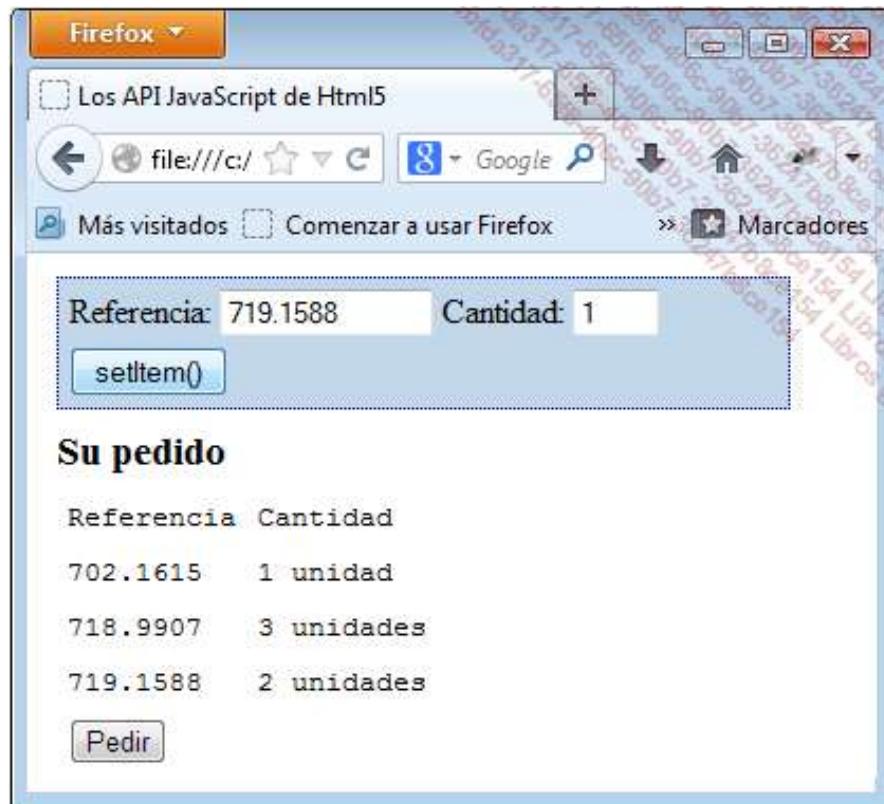
Construyamos un formulario de pedido a partir de un catálogo de venta por correo. Una vez que se ha enviado el pedido, ya no es necesario conservar los datos de este. El objeto `sessionStorage` se adapta perfectamente.

La página inicial:



Para el resto de los ejemplos dedicados a `sessionStorage`, codificamos el pedido siguiente:

- 702.1615 - 1 unidad
- 718.9907 - 3 unidades
- 719.1588 - 2 unidades



Key	Value
702.1615	1 unidad
718.9907	3 unidades
719.1588	2 unidades

Los datos codificados están presentes en **Session Storage - Local Files**.

El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#box { border: 1px dotted navy;
        padding-left: 5px;
        background-color: rgb(195,215,235);
        width: 340px; }
p { margin-top: 5px;
    margin-bottom: 5px; }
td { font-family: monospace;
```

```

padding: 5px;
table { margin-top: 0px;
         border-spacing: 0px; }
.submit { margin-top: 20px; }
</style>
<script type="text/javascript">
function dosetItem() {
var ref = document.getElementById("referencia").value;
var q = document.getElementById("cantidad").value;
sessionStorage.setItem(ref, q);
document.getElementById("referencia").value="";
document.getElementById("cantidad").value="";
editar();
}
function editar() {
var key = "";
var pares = "<tr><td>Referencia</td><td>Cantidad</td></tr>\n";
var i=0;
for (i=0; i<=sessionStorage.length-1; i++) {
key = sessionStorage.key(i);
pares += "<tr><td>" + key + "</td>\n<td>" +
+sessionStorage.getItem(key) + "</td></tr>\n";
}
if (pares == "<tr><td>Referencia</td><td>Cantidad</td></tr>\n") {
pares += "<tr><td colspan=2><center>No hay articulos en el pedido
</center></td></tr>\n";
}
document.getElementById('pares').innerHTML = pares;
}
</script>
</head>
<body onload="editar() ">
<form>
<div id="box">
<p>
<label>Referencia:</label>
<input id="referencia" name="referencia" size="12"
placeholder=
"Referencia">
<label>Cantidad:</label>
<input id="cantidad" name="cantidad" size="2">
</p>
<p>
<input type="button" value="setItem()" onclick="dosetItem()">
</p>
</div>
<h3>Su pedido</h3>
<table id="pares"></table>
<input type="submit" class="submit" value="Pedir">
</form>
</body>
</html>

```

Comentario

```

var ref = document.getElementById("referencia").value;
var q = document.getElementById("cantidad").value;

```

Los variables `ref` y `q` contienen los valores codificados en las respectivas líneas de texto.

```
sessionStorage.setItem(ref, q);
```

A continuación se guardan temporalmente (`sessionStorage.setItem`) con la variable `ref` como clave y la variable `q` como valor.

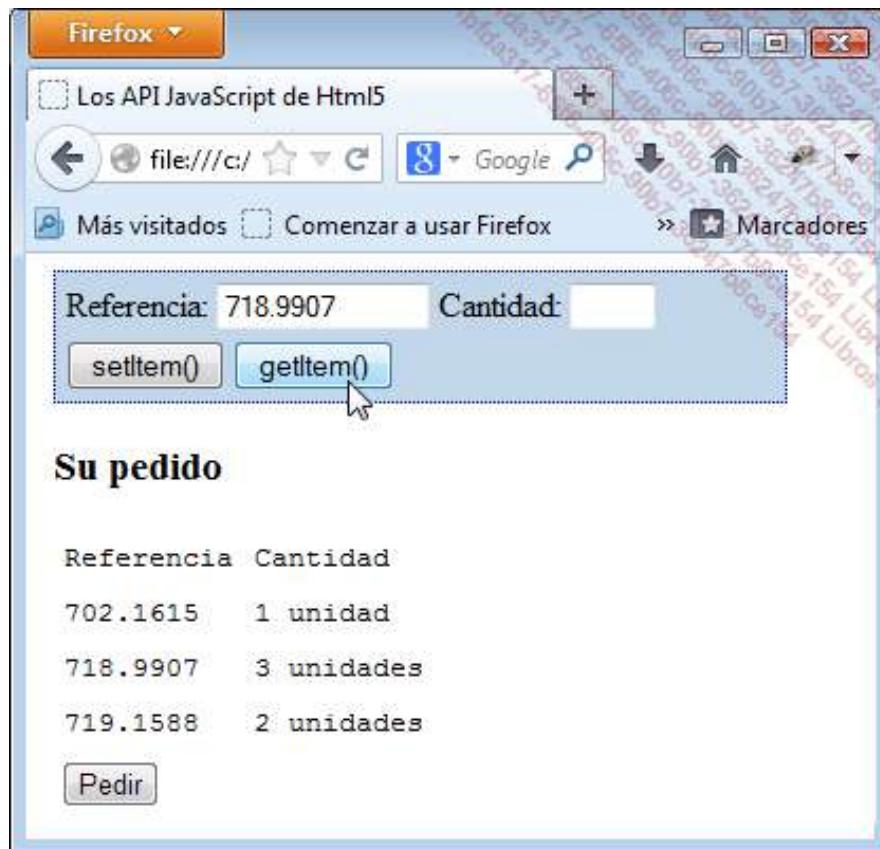
2. Recuperar un valor almacenado

La propiedad `getItem()` permite localizar un valor existente. Su sintaxis es:

```
sessionStorage.getItem(clave);
```

Ejemplo

Queremos localizar la referencia 718.9907 para, por ejemplo, modificar las cantidades.



El resultado:



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#box { border: 1px dotted navy;
        padding-left: 5px;
        background-color: rgb(195,215,235);
        width: 340px; }
p { margin-top: 5px;
    margin-bottom: 5px; }
td { font-family: monospace;
    padding: 5px; }
table { margin-top: 0px;
        border-spacing: 0px; }
.submit { margin-top: 20px; }
</style>
<script type="text/javascript">
function dosetItem() {
var ref = document.getElementById("referencia").value;
var q = document.getElementById("cantidad").value;
sessionStorage.setItem(ref, q);
document.getElementById("referencia").value="";
document.getElementById("cantidad").value="";
editar();
}
function dogetItem() {
var ref = document.getElementById("referencia").value;
document.getElementById("cantidad").value =
sessionStorage.getItem(ref);
editar();
```

```

}

function editar() {
var key = "";
var pares = "<tr><td>Referencia</td><td>Cantidad</td></tr>\n";
var i=0;
for (i=0; i<=sessionStorage.length-1; i++) {
key = sessionStorage.key(i);
pares += "<tr><td>" + key + "</td>\n<td>" +
+sessionStorage.getItem(key) + "</td></tr>\n";
}
if (pares == "<tr><td>Referencia</td><td>Cantidad</td></tr>\n") {
pares += "<tr><td colspan=2><center>No hay artículos en el
pedido</center></td></tr>\n";
}
document.getElementById('pares').innerHTML = pares;
}
</script>
</head>
<body onload="editar ()">
<form>
<div id="box">
<p>
<label>Referencia:</label>
<input id="referencia" name="referencia" size="12"
placeholder= "Referencia">
<label>Cantidad:</label>
<input id="cantidad" name="cantidad" size="2">
</p>
<p>
<input type="button" value="setItem()" onclick="dosetItem()">
<input type="button" value="getItem()" onclick="dogetItem()">
</p>
</div>
<h3>Su pedido</h3>
<table id="pares"></table>
<input type="submit" class="submit" value="Pedir">
</form>
</body>
</html>

```

Comentario

```

var ref = document.getElementById("referencia").value;
document.getElementById("cantidad").value =
sessionStorage.getItem(ref);

```

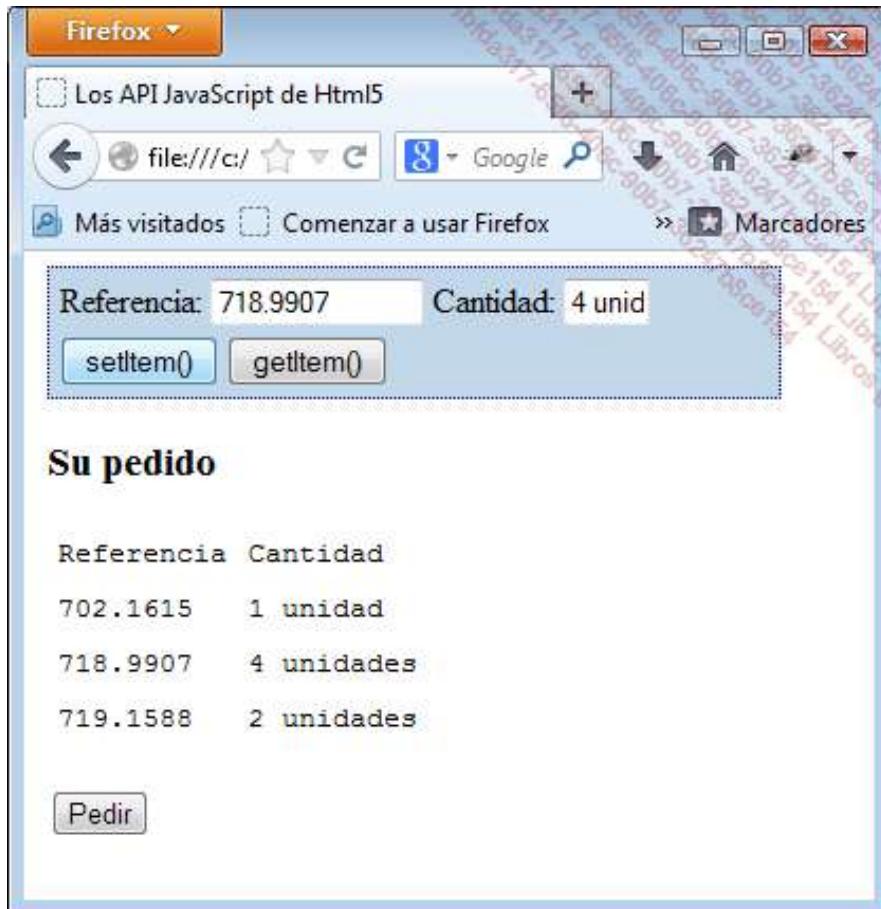
La variable `ref` contiene el valor codificado en la línea de texto dedicada a la referencia. El valor devuelto para la clave `ref` (`sessionStorage.getItem(ref)`) se transmite a la línea de texto relativa a las cantidades e identificada por `cantidad` (`document.getElementById("cantidad")`).

3. Modificar un valor almacenado

Como para `localStorage`, no existe una propiedad específica para modificar un valor almacenado. Es suficiente con llamar a la clave del valor que se va a modificar (`getItem()`), codificar el nuevo valor y almacenar la nueva pareja (`setItem()`). Esto borra el valor antiguo.

Ejemplo

En la referencia 718.9907, vamos a modificar la cantidad para pasar de 3 a 4.



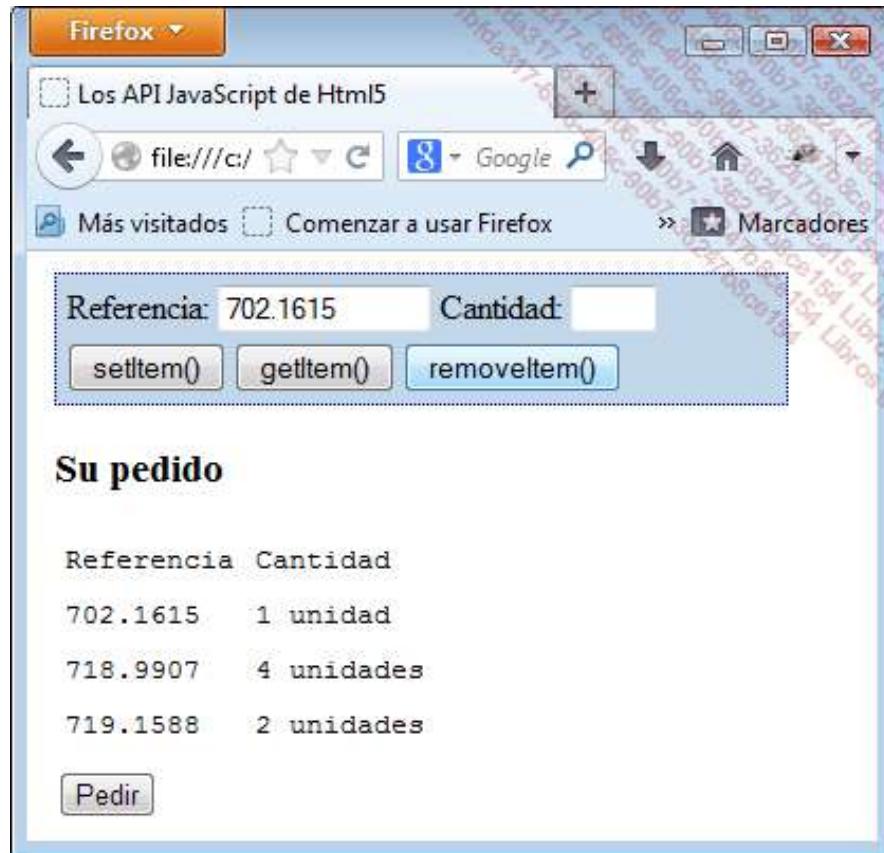
4. Eliminar un valor almacenado

Para eliminar un valor, disponemos de la propiedad `removeItem()`.

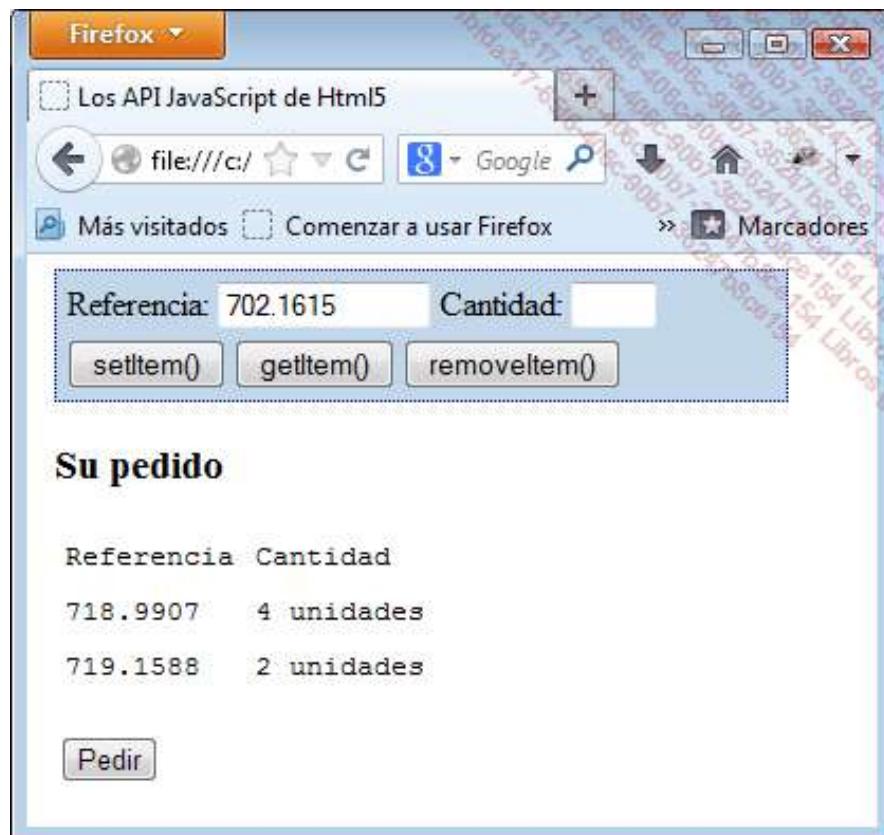
```
localStorage.removeItem(clave);
```

Ejemplo

Finalmente no queremos comprar la referencia 702.1615 y la retiramos de nuestro pedido.



Lo que da como resultado:



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
```

```
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#box { border: 1px dotted navy;
    padding-left: 5px;
    background-color: rgb(195,215,235);
    width: 340px; }
p { margin-top: 5px;
    margin-bottom: 5px; }
td { font-family: monospace;
    padding: 5px; }
table { margin-top: 0px;
    border-spacing: 0px; }
.submit { margin-top: 20px; }
</style>
<script type="text/javascript">
function dosetItem() {
var ref = document.getElementById("referencia").value;
var q = document.getElementById("cantidad").value;
sessionStorage.setItem(ref, q);
document.getElementById("referencia").value="";
document.getElementById("cantidad").value="";
editar();
}
function dogetItem() {
var ref = document.getElementById("referencia").value;
document.getElementById("cantidad").value =
sessionStorage.getItem(ref);
editar();
}
function doremoveItem() {
var ref = document.getElementById("referencia").value;
sessionStorage.removeItem(ref);
editar();
}
function editar() {
var key = "";
var pares = "<tr><td>Referencia</td><td>Cantidad</td></tr>\n";
var i=0;
for (i=0; i<=sessionStorage.length-1; i++) {
key = sessionStorage.key(i);
pares += "<tr><td>" + key + "</td>\n<td>" +
sessionStorage.getItem(key) + "</td></tr>\n";
}
if (pares == "<tr><td>Referencia</td><td>Cantidad</td></tr>\n") {
pares += "<tr><td colspan=2><center>No hay articulos en el pedido
</center></td></tr>\n";
}
document.getElementById('pares').innerHTML = pares;
}
</script>
</head>
<body onload="editar()">
<form>
<div id="box">
<p>
<label>Referencia:</label>
<input id="referencia" name="referencia" size="12"
placeholder="Referencia">
<label>Cantidad:</label>
<input id="cantidad" name="cantidad" size="2">
</p>
<p>
<input type="button" value="setItem()" onclick="dosetItem()">
<input type="button" value="getItem()" onclick="dogetItem()">
<input type="button" value="removeItem()">

```

```

    onclick="doremoveItem()" >
</p>
</div>
<h3>Su pedido</h3>
<table id="pares"></table>
<input type="submit" class="submit" value="Pedir">
</form>
</body>
</html>

```

Comentario

```

var ref = document.getElementById("referencia").value;
sessionStorage.removeItem(ref);

```

La variable `ref` contiene lo que se ha codificado en la línea de texto que recibe las referencias. Después se envía como clave a la propiedad `removeItem()`.

5. Eliminar todos los valores almacenados

Para eliminar todos los pares clave/valor, tenemos a nuestra disposición la propiedad `clear()`. En nuestro ejemplo, se corresponde con un botón **Clear**.

```
localStorage.clear();
```



La página se convierte en:



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#box { border: 1px dotted navy;
    padding-left: 5px;
    background-color: rgb(195,215,235);
    width: 340px; }
p { margin-top: 5px;
    margin-bottom: 5px; }
td { font-family: monospace;
    padding: 5px; }
table { margin-top: 0px;
    border-spacing: 0px; }
.submit { margin-top: 20px; }

</style>
<script type="text/javascript">
function dosetItem() {
var ref = document.getElementById("referencia").value;
var q = document.getElementById("cantidad").value;
sessionStorage.setItem(ref, q);
document.getElementById("referencia").value="";
document.getElementById("cantidad").value="";
editar();
}

function dogetItem() {
var ref = document.getElementById("referencia").value;
document.getElementById("cantidad").value =
sessionStorage.getItem(ref);
editar();
}
function doremoveItem() {
```

```

var ref = document.getElementById("referencia").value;
sessionStorage.removeItem(ref);
editar();
}
function doClear() {
sessionStorage.clear();
editar();
}
function editar() {
var key = "";
var pares = "<tr><td>Referencia</td><td>Cantidad</td></tr>\n";
var i=0;
for (i=0; i<=sessionStorage.length-1; i++) {
key = sessionStorage.key(i);
pares += "<tr><td>" + key + "</td>\n<td>" +
+sessionStorage.getItem(key) + "</td></tr>\n";
}
if (pares == "<tr><td>Referencia</td><td>Cantidad</td></tr>\n") {
pares += "<tr><td colspan=2><center>No hay artículos en el pedido
</center></td></tr>\n";
}
document.getElementById('pares').innerHTML = pares;
}
</script>
</head>
<body onload="editar()">
<form>
<div id="box">
<p>
<label>Referencia:</label>
<input id="referencia" name="referencia" size="12"
placeholder="Referencia">
<label>Cantidad:</label>
<input id="cantidad" name="cantidad" size="2">
</p>
<p>
<input type="button" value="setItem()" onclick="dosetItem()">
<input type="button" value="getItem()" onclick="dogetitem()">
<input type="button" value="removeItem()"
onclick="doremoveitem()">
<input type="button" value="clear()" onclick="doClear()">
</p>
</div>
<h3>Su pedido</h3>
<table id="pares"></table>
<input type="submit" class="submit" value="Pedir">
</form>
</body>
</html>

```

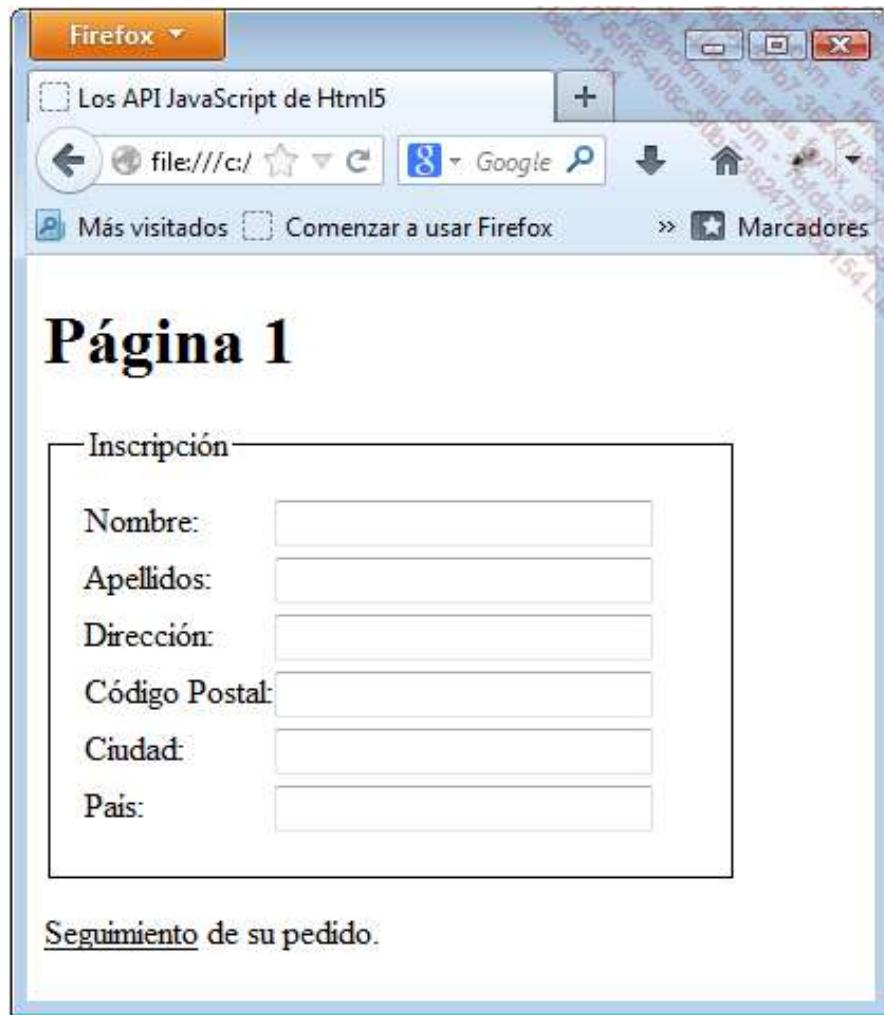
6. Aplicación final

Algunas veces, pasar un valor de una página a otra página de un sitio Web es una pesadilla que requiere recurrir a trucos o al envío de estos datos a través del servidor.

El objeto sessionStorage aporta una solución elegante a este problema. De hecho, es suficiente con almacenar en local estos datos y llamarlos cuando sea necesario. No hay nada más sencillo.

Ejemplo

Pensemos en un formulario inicial (pagina1.htm).



El usuario completa este formulario y los datos se almacenan en el espacio SessionStorage.

Firefox

Los API JavaScript de Html5

file:///c:/

Más visitados Comenzar a usar Firefox Marcadores

Página 1

Inscripción

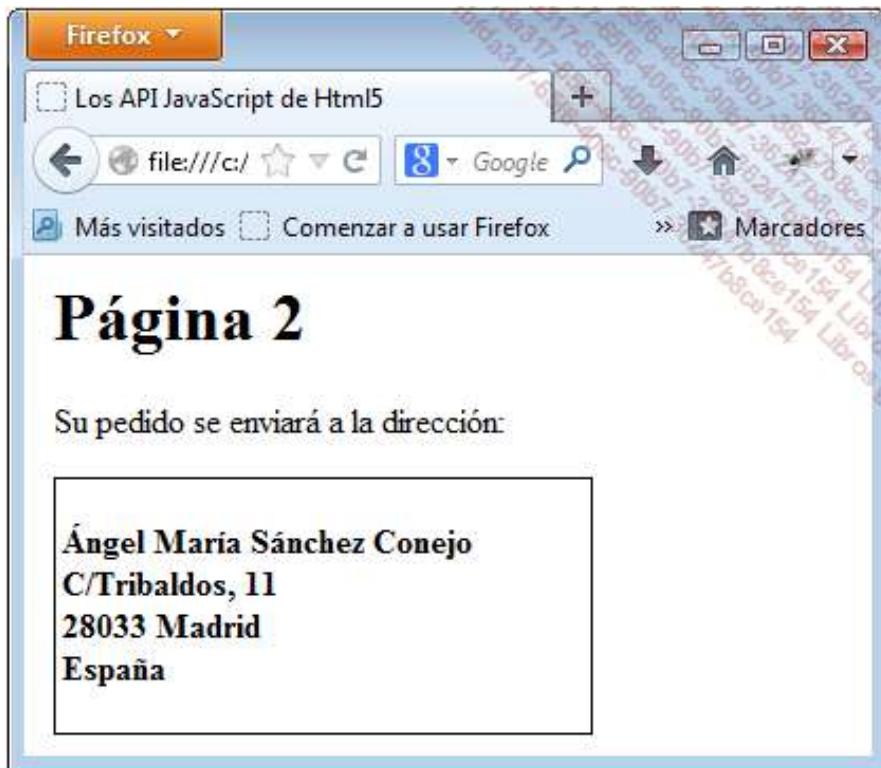
Nombre:	Ángel María
Apellidos:	Sánchez Conejo
Dirección:	C/Tribaldos, 11
Código Postal:	28033
Ciudad:	Madrid
País:	España

Seguimiento de su pedido.

Key	Value
apellidos	Sánchez Conejo
ciudad	Madrid
cp	28033
direccion	C/Tribaldos, 11
nombre	Ángel María
país	España

The screenshot shows the Firefox developer tools Network tab. On the left, there's a tree view of storage types: Frames, Web SQL, IndexedDB, Local Storage, Session Storage, Cookies, and Application Cache. Under Local Storage and Session Storage, there are entries for 'file://'. The Session Storage entry for 'file://' is selected, showing a table of stored data with columns 'Key' and 'Value'. The data listed is identical to the form data shown in the first screenshot.

En otra página del sitio Web (pagina2.htm), es suficiente con llamar a los valores que hay en sessionStorage.



Código de pagina1.htm

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
a { color: black; }
form.css fieldset { padding: 1em;
                     width: 290px;
                     border: 1px solid black; }
form.css label { display: inline;
                  float: left;
                  width: 90px; }
input { margin-bottom: 5px; }
</style>
<script type="text/javascript">
function set_item() {
var nombre = "nombre";
var dato1 = document.getElementById("f1").value
sessionStorage.setItem(nombre, dato1);
var apellidos = "apellidos";
var dato2 = document.getElementById("f2").value
sessionStorage.setItem(apellidos, dato2);
var direccion = "dirección";
var dato3 = document.getElementById("f3").value
sessionStorage.setItem(direccion, dato3);
var cp = "cp";
var dato4 = document.getElementById("f4").value
sessionStorage.setItem(cp, dato4);
var ciudad = "ciudad";
var dato5 = document.getElementById("f5").value
sessionStorage.setItem(ciudad, dato5);
var pais = "pais";
var dato6 = document.getElementById("f6").value
sessionStorage.setItem(pais, dato6);
}

```

```

</script>
</head>
<body>
<h1>Página 1</h1>
<form class="css">
<fieldset>
<legend>Inscripción</legend>
<label>Nombre: </label>
<input id="f1" size="25" required><br>
<label>Apellidos: </label>
<input id="f2" size="25" required><br>
<label>Dirección: </label>
<input id="f3" size="25" required><br>
<label>Código Postal: </label>
<input id="f4" size="25" required><br>
<label>Ciudad: </label>
<input id="f5" size="25" required><br>
<label>País: </label>
<input id="f6" size="25" required><br>
</fieldset>
</form>
<p><a href="pagina2.htm" onclick="set_item()">Seguimiento</a>
de su pedido.</p>
</body>
</html>

```

Comentario

Hacemos los comentarios siguientes:

```

var nombre = "nombre";
var dato1 = document.getElementById("f1").value
sessionStorage.setItem(nombre, dato1);

```

La clave está contenida en la variable var nombre = "nombre". El valor es el de la línea de texto (var dato1 = document.getElementById("f1").value). Se almacena la pareja clave/valor (sessionStorage.setItem(nombre, dato1)). Es igual para el resto de las entradas.

Código de la pagina2.htm

```

<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
div { border: 1px solid black;
      font-size:16px;
      font-weight: bold;
      width: 250px;
      padding-left: 3px; }
</style>
<script type="text/javascript">
function get_item() {
document.getElementById("f1").innerHTML =
sessionStorage.getItem("nombre");
document.getElementById("f2").innerHTML =
sessionStorage.getItem("apellidos");
document.getElementById("f3").innerHTML =
sessionStorage.getItem("dirección");
document.getElementById("f4").innerHTML =
sessionStorage.getItem("cp");
document.getElementById("f5").innerHTML =

```

```
sessionStorage.getItem("ciudad");
document.getElementById("f6").innerHTML =
sessionStorage.getItem("pais");
}
</script>
</head>
<body onload="get_item()">
<h1>Página 2</h1>
<form>
<p>Su pedido se enviará a la dirección:<p>
<div>
<span id="f1"></span> <span id="f2"></span><br>
<span id="f3"></span><br>
<span id="f4"></span> <span id="f5"></span><br>
<span id="f6"></span>
</div>
</form>
</body>
</html>
```

Comentario

sessionStorage.getItem("nombre") recupera el valor almacenado y lo mismo para los otros campos.

Los eventos del objeto Storage

A lo largo de los diferentes borradores (*draft*) de la especificación Web Storage, aparecieron los eventos relacionados con el almacenamiento de datos. Estos eventos de almacenamiento (*storage events*) se producen con cada modificación del objeto Storage.

El objeto storage events tiene las siguientes propiedades:

key	Representa la clave que se modifica.
oldValue	Representa el valor antiguo de la clave que se modifica.
newValue	Representa el nuevo valor de la clave que se modifica.
url	Representa la URL de la página cuya clave se ha modificado.
storageArea	Representa el objeto de almacenamiento que se ha modificado.

Estos eventos, que aparecieron más tarde (y algunas veces de manera parcial), solo se usan en:

- Internet Explorer 9.
- Firefox 5.
- Google Chrome 12.
- Opera 10.5.

Para tener acceso a estas propiedades del evento de almacenamiento, es necesario usar los manejadores de eventos DOM 2 (AddEventListener); por ejemplo:

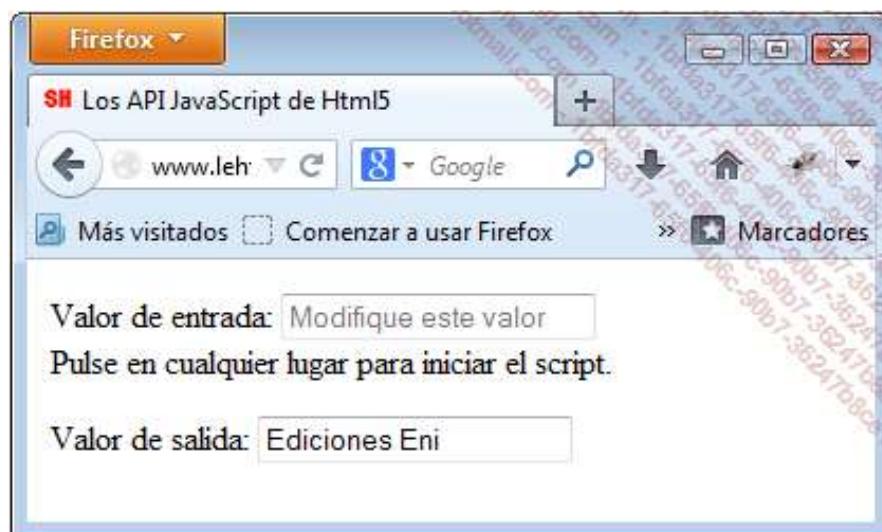
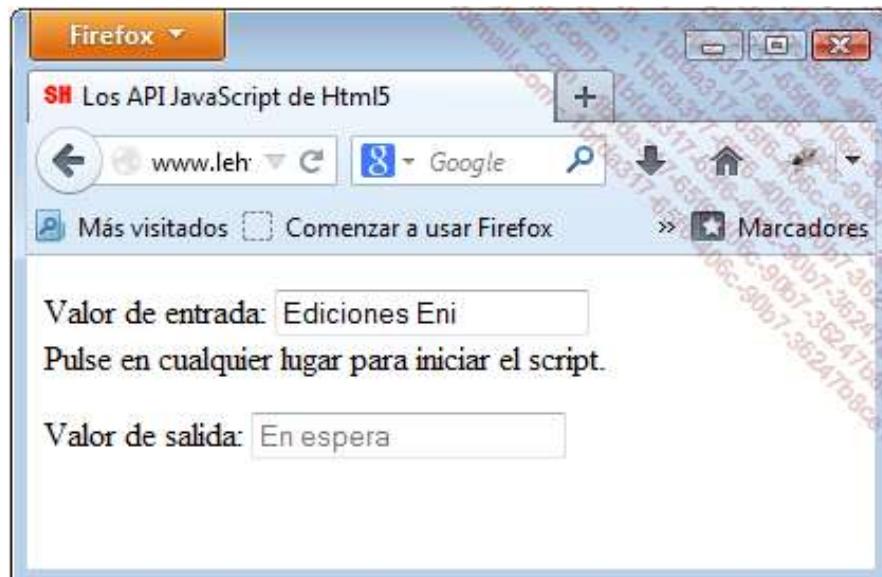
```
window.addEventListener('storage', function(event) {  
  console.log('El valor de ' + event.key + ' se ha cambiado de ' +  
    event.oldValue + ' a ' + event.newValue);  
}, false);
```

Observe que, para cada almacenamiento de datos, el evento de almacenamiento afecta a otra instancia del navegador o a otra pestaña.

Ejemplo

Supongamos una página muy sencilla con una zona de entrada y una de salida. Esta página se debe mostrar al mismo tiempo en dos pestañas o ventanas diferentes del navegador. Vamos a usar los eventos de almacenamiento para que aparezca en la página de la segunda pestaña la codificación que hay en la página de la primera pestaña.

Para el correcto funcionamiento de este script, debe estar conectado o en un servidor local.



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<script type="text/javascript">
function init(){
var input = document.getElementById('data');
var output = document.getElementById('evento_de_salida');
window.addEventListener('storage', function (event) {
if (event.key == 'test') {
output.value = event.newValue;
}
});
input.addEventListener('blur', function () {
localStorage.setItem('test', input.value);
});
}
</script>
</head>
<body onload="init()">
<p>
Valor de entrada: <input type="text" id="data"
```

```
placeholder="Modifique este valor " value=""><br>
Pulse en cualquier lugar para iniciar el script.
</p>
<p>
Valor de salida: <input type="text" id="evento_de_salida"
placeholder="En espera" value="">
</p>
</body>
</html>
```

Comentario

```
var input = document.getElementById('data');
var output = document.getElementById('evento_de_salida');
```

Las variables `input` y `output` apuntan respectivamente a las líneas de texto de entrada y salida.

```
input.addEventListener('blur', function () {
localStorage.setItem('test', input.value);
});
```

Añadimos un manejador de eventos DOM 2 a la línea de texto de entrada (`input.addEventListener`). Este, cuando se pierde el foco (`blur`), almacena de manera permanente (`localStorage.setItem`) la clave `test` con valor el contenido de la línea de texto de entrada (`input.value`).

```
window.addEventListener('storage', function (event) {
if (event.key == 'test') {
output.value = event.newValue;
}
});
```

El script añade un manejador de eventos DOM 2 (`addEventListener` al objeto `window`). Cuando se produce una modificación del objeto `storage`, este evento se pasa como argumento de una función (`function (event)`). Este comprueba en primer lugar si la clave corresponde a `test` (`if`). En caso afirmativo, la zona de texto de salida (`output.value`) toma como nuevo valor (`event.newValue`) el valor inicial codificado en la línea de texto de entrada.

Presentación y objetivos

Almacenar datos de manera temporal o permanente es una ventaja significativa en el lenguaje Html. Pero imagino que algunos desarrolladores no estarán satisfechos y desearán un sistema todavía más evolucionado. Cuando hay que gestionar una cantidad importante de datos, sería interesante tener una base de datos debidamente estructurada, con un acceso SQL. Una consulta de un catálogo de venta, un listado de clientes o una gestión de almacenes ahora es posible a partir de una aplicación Web. Con Html5, quedan muy lejos las páginas estáticas de los años noventa y cada vez más cerca las aplicaciones de escritorio.

El API Web SQL Database permite almacenar en el navegador bases de datos y tratar estos datos con SQLite, directamente instalado en el navegador.

Esta última frase merece numerosos comentarios:

- Cuando la base de datos se carga en el navegador, esta permanece presente en el navegador y se puede consultar conectado o desconectado (*online* u *offline*).
- SQLite está instalado en el navegador, es decir, en el lado cliente. Esto permite salir del esquema clásico de llamadas al servidor y después al sistema de gestión de la base de datos (SGBD). La ventaja consiste en un proceso rápido de consulta y tratamiento de los datos y de llamadas económicas al servidor que siempre ralentizan la fluidez de las aplicaciones. A riesgo de repetirnos, la llamada al servidor es el enemigo de las API JavaScript de Html5. Esto es muy conveniente para las aplicaciones Web para Smartphone.
- El proceso de este API es asíncrono. Esto contribuye a la usabilidad de las aplicaciones, porque las consultas SQL se hacen sin molestar o ralentizar el proceso normal de la aplicación.
- SQLite es un sistema de gestión de base de datos transaccional. Esto tendrá una influencia en el código de los scripts. Volveremos a tratarlo cuando abordemos los ejemplos de este capítulo.
- El código usado estará basado en JavaScript, en el que se anidarán algunas sentencias de SQLite.
- Para los amantes de la precisión, se utiliza la versión 3.1.19 de SQLite.
- Este API Web SQL Database no forma parte de Html5, sino que es una especificación separada.

Este API hace que sean necesarias algunas nociones de SQL. Para los desarrolladores y los más expertos, resultará sorprendente que solo sean necesarias algunas nociones de SQL (pienso fundamentalmente en MySQL). Por lo tanto, no se sentirán sobrepasados porque las instrucciones están muy cerca de las que usan en MySQL. Los demás deberán adquirir algunas nociones de SQL. Los tutoriales son abundantes y Google le ayudará a encontrar algunos. Por ejemplo:

- <http://sqlite.org>
- <http://www.tutorialspoint.com/sql/index.htm>
- <http://www.1keydata.com/es/sql/>

El API Web SQL Database de Html5 es ideal para las aplicaciones móviles que necesitan el soporte de una base de datos compleja. Los datos se guardan de manera local incluso cuando el aparato se apaga. La presencia de SQLite en el navegador garantiza un acceso rápido a los datos sin tener que hacer llamadas al servidor.

A nivel de la seguridad, solo podrá acceder a la base de datos el dominio que ha generado la aplicación. De esta manera, los datos almacenados en el sitio Web A no son visibles ni accesibles por el sitio Web B. Esto impide, por ejemplo, que una etiqueta <iframe> de un sitio Web sea un banner publicitario que permita el acceso a los datos guardados localmente por este sitio Web. Este sistema no protege de algunos ataques que se describen en la especificación, como los ataques por usurpación de DNS, la compartición del nombre del host o los ataques por XSS (*cross site scripting*). A pesar de la enumeración de estos riesgos, conviene insistir en el hecho de que no son específicos de Html5, sino que es inherente a la arquitectura de la Web desde sus orígenes. Para más información sobre la seguridad de las bases de datos que se basan en SQL,

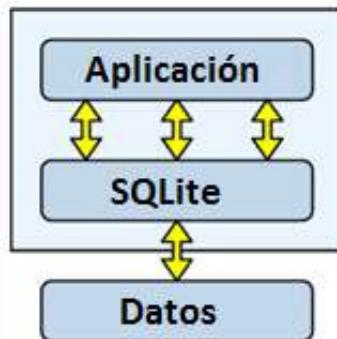
consulte: www.code.google.com/p/html5security/wiki/WebSQLDatabaseSecurity

En el terreno de la gestión de las bases de datos, seguramente conoce MySQL o PostgreSQL. Parece útil presentar SQLite más detalladamente.

SQLite es una librería escrita en C que ofrece un motor de base de datos relacional, al que se accede utilizando el lenguaje SQL.

La principal particularidad de SQLite es que se puede integrar directamente en la aplicación que utiliza su librería sin pasar por el esquema tradicional cliente-servidor.

Modelo integrado en SQLite



Se distingue de esta manera de la mayoría de sistemas de gestión de bases de datos, como MySQL, que deben hacer llamadas a un servidor externo.

Modelo Cliente-Servidor



Esto hace que la consulta a la base de datos con SQLite sea más rápida, porque evita el acceso al servidor.

Además de su tamaño extremadamente reducido (300 KB), SQLite tiene una excelente portabilidad porque los archivos de gestión de la base de datos son totalmente independientes del sistema operativo y de la arquitectura en la que se utilizan. Por tanto, no es sorprendente encontrarlo en dispositivos tan diversos como misiles o aviones no tripulados, pasando por los Smartphone.

SQLite es Open Source y puede ser el sistema de gestión de base de datos más utilizado en el mundo. SQLite tiene clientes tan importantes como:

- Adobe, que utiliza SQLite para su Photoshop Lightroom. También forma parte de Adobe Air.
- Apple, que retoma SQLite para numerosas funcionalidades de su sistema Mac OS X, como Apple Mail, Safari y Aperture. SQLite también está presente en el software del iPhone y los iPod.
- Google, que utiliza SQLite en su escritorio para Mac, en Google Gears, en su sistema operativo Android y probablemente en otras aplicaciones ignoradas por el público.

- McAfee, que incluye SQLite en sus programas antivirus.
- PHP, que usa al mismo tiempo SQLite2 y SQLite3 en sus librerías de manera estándar.
- Airbus, que confirma que SQLite se usa en el software de vuelo para los A350 XWB de su gama de aviones.
- Etc.

SQLite recupera las funcionalidades principales de gestión de base de datos. Pero probablemente, a causa de su reducido tamaño, SQLite no está exento de algunos problemas o lagunas (dificultades cuando un número importante de clientes acceden a la base, ausencia de algunas funcionalidades SQL evolucionadas). De una manera general, nos limitaremos a bases de datos de un tamaño razonable y capaces de dar servicio a un número limitado de clientes.

Esta presentación de SQLite se extrae del excelente artículo que le dedica la Wikipedia. Los lectores interesados pueden consultar el artículo completo y muy bien documentado en la dirección <http://es.wikipedia.org/wiki/SQLite>.

Todos los ejercicios y prácticas se pueden hacer directamente con los archivos situados, por ejemplo, en el escritorio o en un directorio dedicado, sin que sea necesario pasar por un servidor local o tener que poner en línea los archivos.

Disponibilidad del API

Este sistema de base de datos con SQLite se implementa poco en los navegadores de escritorio. Solo se ha adoptado en:

- Safari 4+.
- Chrome 5+.
- Opera 10.5+.

Firefox ha preferido el sistema Indexed Databases (ver el capítulo El API Indexed Database).

Internet Explorer 9 no utiliza ningún proceso de bases de datos a partir de los API JavaScript de Html5, pero Internet Explorer 10 adopta el método Indexed Databases.

Por el contrario, este API ha tenido mucha aceptación en los Smartphone y tabletas.

- iOS Safari 3.2+.
- Opera Móvil 11+.
- Android 2.0+.

Como este API es poco compatible, es prudente verificar su disponibilidad.

Para crear un objeto de base de datos, utilizamos el método `openDatabase()` (ver la siguiente sección: Crear una base de datos). El objetivo del test de compatibilidad es probar la existencia de esta propiedad del objeto JavaScript `window`. Si el valor devuelto es `true`, el navegador soporta las bases de datos SQL y es posible usarlo.

```
if (!window.openDatabase) {  
    alert("Su navegador no soporta la creación de bases  
    de datos SQL");  
}  
else {  
    alert("Su navegador acepta la creación de bases de datos  
    SQL");  
}
```

Ejemplo

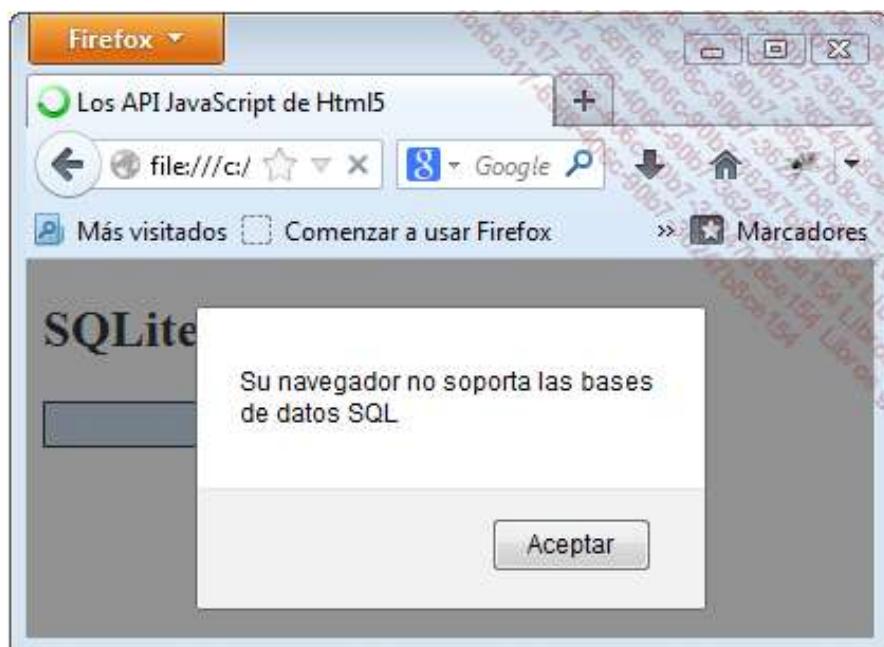
```
<!DOCTYPE html>  
<html lang="es">  
<head>  
<title>Los API JavaScript de Html5</title>  
<meta charset=utf-8>  
<style type="text/css">  
#box { width: 270px;  
        border: 1px solid black;  
        background-color: rgb(195,215,235);  
        padding-left: 3px;}  
</style>  
<script type="text/javascript">  
function init(){  
if (!window.openDatabase) {  
    alert("Su navegador no soporta las bases de datos  
    SQL");  
}  
else {  
msg = "Se tratan las bases de datos SQL ";  
document.querySelector('#box').innerHTML = msg;  
}  
}  
</script>
```

```
</head>
<body onload="init();">
<h2>SQLite Database</h2>
<div id="box">&nbsp;</div>
</body>
</html>
```

En Safari:



En Firefox:



Crear una base de datos

Para crear el objeto base de datos, el API llama al método openDatabase.

La sintaxis es:

```
openDatabase ('nombre de la base de datos', 'número de la versión de la base de datos', 'descripción de la base de datos', 'dimensión de la base de datos', 'funcionalidades de callback');
```

Vamos a revisar estos cinco argumentos.

Nombre de la base datos

Es el nombre de la base de datos que se utilizará en las siguientes líneas del script de la aplicación. Este dato es obligatorio.

Número de versión

Este número de versión de la base de datos es un dato obligatorio, que será necesario mencionar al inicio de la base de datos. Aunque esto va más allá de los objetivos de este libro, la información disponible en la Red indica grandes dificultades para migrar de una versión a otra del mismo. El artículo disponible en la página <http://blog.maxaller.name/2010/03/html5-web-sql-database-intro-to-versioning-and-migrations/> ilustra muy bien el problema.

Descripción de la base de datos

Como su propio nombre indica, el desarrollador puede describir aquí la función de la base de datos. Esta descripción no se utilizará para nada en el código y es opcional.

Dimensión de la base de datos

Fijamos el tamaño que pensamos que tendrá la base de datos. Recordamos el límite, que se fija arbitrariamente en 5 MB por dominio. Por dominio, es posible crear tantas bases de datos como quiera. Veremos más adelante en este capítulo que algunos navegadores en sus últimas versiones permiten aumentar este límite.

Funciones de callback

Estas funciones de callback se pueden prever en caso de éxito o error durante la creación de la base de datos.

Por ejemplo, en caso de error:

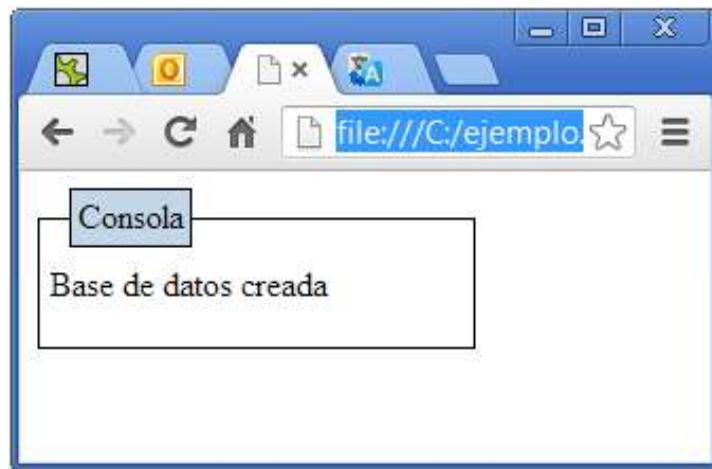
```
webdb.onError = function(tx, e) {
  alert('Error: ' + e.message );
}
```

O en caso de éxito:

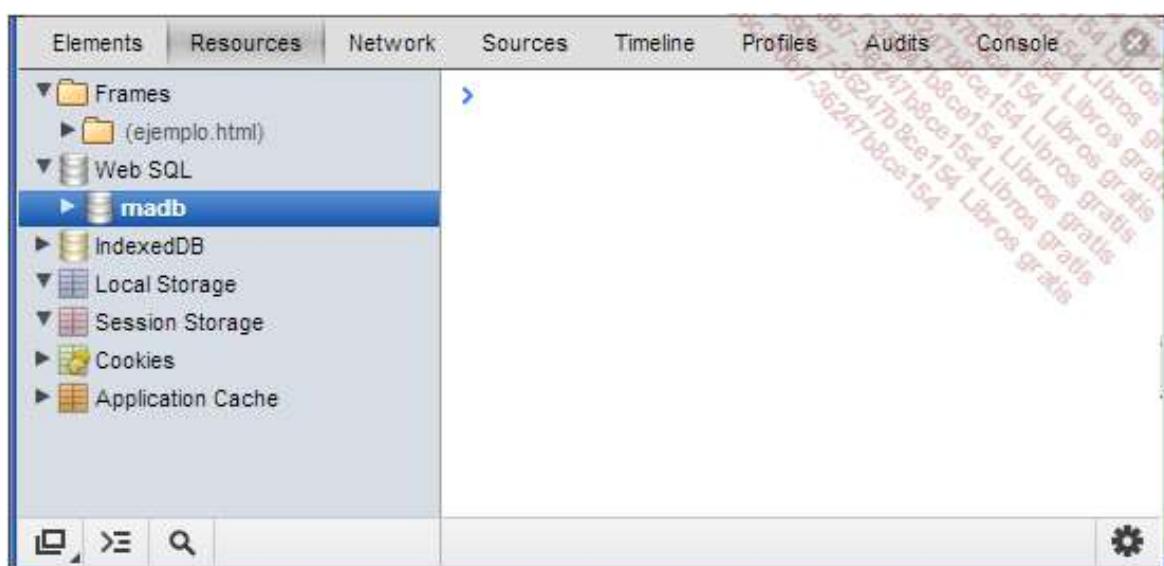
```
webdb.onSuccess = function(tx, r) {
  //
  // Sigue el script
  //
}
```

Ejemplo

Creamos una base de datos cuyo nombre es madb, la versión es 1.0 y el tamaño 2 MB.



Visto desde el Web Inspector de Safari ([Ctrl][Alt] I):



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#estado { position: absolute;
            border: 1px solid black;
            width: 200px;
            padding-left: 5px;
            margin-top: 14px;
            z-index: 1;}
#console { position: absolute;
            border: 1px solid black;
            padding: 3px;
            margin-left: 15px;
            margin-top: 3px;
            width: 50px;
            background-color: rgb(195,215,235);
            z-index: 2;}
</style>
<script type="text/javascript">
function init(){
var db = openDatabase('madb', '1.0', 'Test DB', 2 * 1024 * 1024);
```

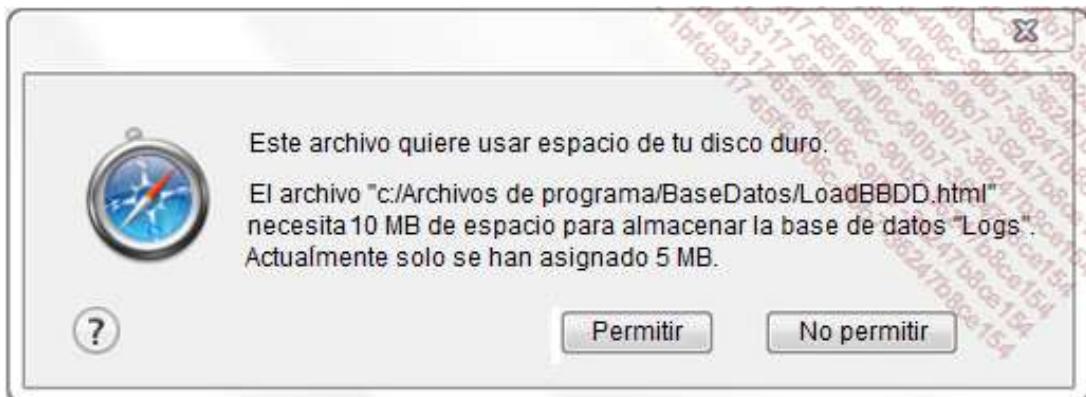
```
var msg;
msg = 'Base de datos creada<br>';
document.querySelector('#estado').innerHTML += msg;
}
</script>
</head>
<body onload="init()">
<div id="console">Consola</div>
<div id="estado"><br></div>
</body>
</html>
```

Comentario

```
var db = openDatabase('mdb', '1.0', 'Test DB', 2 * 1024 * 1024);
```

Hemos creado la base de datos (`openDatabase()`) llamada mdb, cuya versión es 1.0, la descripción es Test DB y el tamaño 2 MB($2 * 1024 * 1024$).

En el momento de escribir este libro, solo Safari invita al usuario a asignar más espacio cuando la base de datos que la página intenta crear sobrepasa los 5 MB asignados por defecto. El cuadro de diálogo que se muestra en la siguiente captura de pantalla pide la autorización del usuario para asignar un tamaño mayor a la base de datos (5, 10, 50, 100 y 500 MB).



Crear una tabla de datos

Para crear la tabla, utilizaremos la sintaxis SQL:

```
"CREATE TABLE IF NOT EXISTS nombre_de_la_tabla (id_clave_única, nombre_de_la_columna)"
```

Es decir, crear la tabla `nombre_de_la_tabla` si no existe.

Observe que para la construcción de scripts, SQLite funciona con un sistema llamado de transacción. La ventaja de las transacciones consiste en que, si la transacción falla (error SQL o de código), no se hace ninguna modificación.

Un sistema de gestión de bases de datos tiene un índice (id) llamado clave única. Su descripción puede tener diferentes formas según el grado de sofisticación que el desarrollador le quiera asignar. Podemos encontrar:

- id UNIQUE.
- id INTEGER NOT NULL PRIMARY KEY.
- id INTEGER PRIMARY KEY ASC.
- id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT.
- etc.

Observe que SQLite permite claves primarias con valor NULL.

Se trata de un no-respeto a la norma que se podría solucionar en próximas versiones. Por tanto, se aconseja añadir la restricción NOT NULL a la declaración de una clave primaria.

Ejemplo

Creamos la tabla LOGS con dos columnas: una con la clave primaria y otra llamada log.





El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#estado { position: absolute;
            border: 1px solid black;
            width: 200px;
            padding-left: 5px;
            margin-top: 14px;
            z-index: 1;}
#console { position: absolute;
            border: 1px solid black;
            padding: 3px;
            margin-left: 15px;
            margin-top: 3px;
            width: 50px;
            background-color: rgb(195,215,235);
            z-index: 2;}
</style>
<script type="text/javascript">
function init(){
var db = openDatabase('madb', '1.0', 'Test DB', 2 * 1024 * 1024);
var msg;
msg = 'Base de datos creada<br>';
document.querySelector('#estado').innerHTML += msg;
db.transaction(function (tx) {
tx.executeSql('CREATE TABLE IF NOT EXISTS LOGS (id unique, log)');
msg = 'Tabla creada<br>';
document.querySelector('#estado').innerHTML += msg;
});
}
</script>
</head>
<body onload="init()">
<div id="console">Consola</div>
<div id="estado"><br></div>
</body>
</html>
```

Comentario

```
db.transaction(function (tx) {
  tx.executeSql('CREATE TABLE IF NOT EXISTS LOGS (id unique, log)');
  msg = 'Tabla creada<br>';
  document.querySelector('#estado').innerHTML += msg;
});
```

El script lanza una transacción para la variable db que contiene la base de datos (db.transaction). Cede el testigo a la ejecución del SQL (executeSql) que solicita la creación de la tabla LOGS si no existe (CREATE TABLE IF NOT EXISTS). Esta tabla tiene dos columnas; la clave primaria (id unique) y la columna log.

Añadir, seleccionar y visualizar datos

1. Método estático

Para insertar datos, la sintaxis SQL es:

```
"INSERT INTO nombre_de_la_tabla (columna_1, columna_2)  
VALUES (valor_para_la_columna_1, valor_para_la_columna_2)"
```

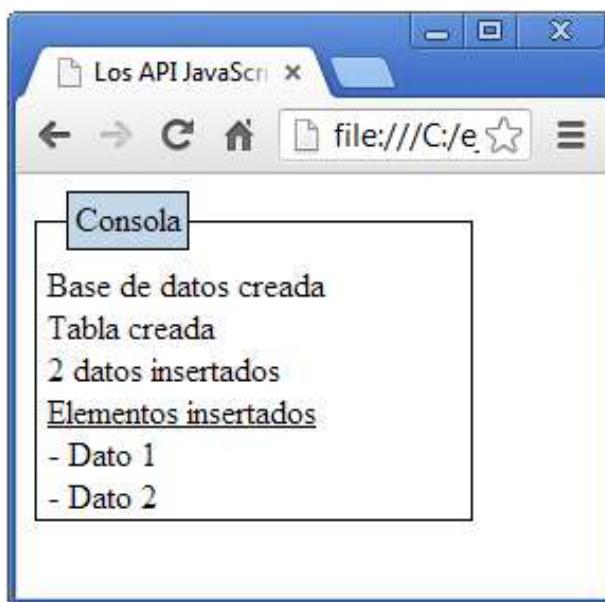
Para seleccionar elementos de la tabla:

```
"SELECT * FROM nombre_de_la_tabla"
```

Para seleccionar todos los elementos (*) de la tabla.

Ejemplo

Añadimos dos datos (Dato 1 y Dato 2) a la tabla LOGS y los visualizamos en la consola.



The screenshot shows the "Network" tab of the Chrome DevTools. On the left, there is a tree view of network resources, including "Frames", "(ejemplo.html)", "Web SQL", "madb", and "LOGS". The "LOGS" item is selected and highlighted in blue. On the right, there is a table with two rows of data. The table has two columns: "id" and "log". The first row has "id" value 1 and "log" value "Dato 1". The second row has "id" value 2 and "log" value "Dato 2".

1	Dato 1
2	Dato 2

El código

```

<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#estado { position: absolute;
            border: 1px solid black;
            width: 200px;
            padding-left: 5px;
            margin-top: 14px;
            z-index: 1;}
#console { position: absolute;
            border: 1px solid black;
            padding: 3px;
            margin-left: 15px;
            margin-top: 3px;
            width: 50px;
            background-color: rgb(195,215,235);
            z-index: 2;}
</style>
<script type="text/javascript">
function init(){
var db = openDatabase('madb', '1.0', 'Test DB', 2 * 1024 * 1024);
var msg;
msg = 'Base de datos creada<br>';
document.querySelector('#estado').innerHTML += msg;
db.transaction(function (tx) {
tx.executeSql('CREATE TABLE IF NOT EXISTS LOGS (id unique, log)');
msg = 'Tabla creada<br>';
document.querySelector('#estado').innerHTML += msg;
});
db.transaction(function (tx) {
tx.executeSql('INSERT INTO LOGS (id, log) VALUES (1, "Dato 1")');
tx.executeSql('INSERT INTO LOGS (id, log) VALUES (2, "Dato 2")');
msg = '2 datos insertados<br>';
document.querySelector('#estado').innerHTML += msg;
});
db.transaction(function (tx) {
tx.executeSql('SELECT * FROM LOGS', [], function (tx, results) {
var len = results.rows.length, i;
document.querySelector('#estado').innerHTML +=
"<u>Elementos insertados</u><br>";
for (i = 0; i < len; i++) {
msg = "- " + results.rows.item(i).log + "</br>";
document.querySelector('#estado').innerHTML += msg;
}
}, null);
});
}
</script>
</head>
<body onload="init()">
<div id="console">Consola</div>
<div id="estado"><br></div>
</body>
</html>

```

Comentario

```

db.transaction(function (tx) {
tx.executeSql('INSERT INTO LOGS (id, log) VALUES (1, "Dato 1")');
tx.executeSql('INSERT INTO LOGS (id, log) VALUES (2, "Dato 2")');

```

```
msg = '2 datos insertados<br>';
document.querySelector('#estado').innerHTML += msg;
});
```

La transacción para la variable db solicita al SQL (executeSql) la inserción en la tabla LOGS(INSERT INTO LOGS), con las columnas id y log, de los siguientes valores (VALUES (1, "Dato 1") y VALUES (2, "Dato 2")).

```
db.transaction(function (tx) {
  tx.executeSql('SELECT * FROM LOGS', [], function (tx, results) {
    var len = results.rows.length, i;
    document.querySelector('#estado').innerHTML +=
      "<u>Elementos insertados</u><br>";
    for (i = 0; i < len; i++) {
      msg = "- " + results.rows.item(i).log + "</br>";
      document.querySelector('#estado').innerHTML += msg;
    }
  }, null);
});
```

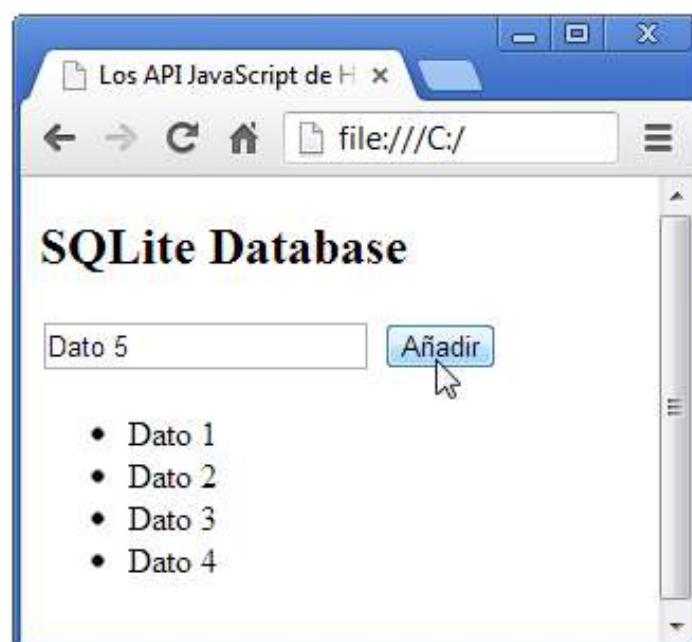
Para poder visualizar los datos insertados, es necesario seleccionarlos en la tabla. La transacción (db.transaction) solicita al SQL (executeSql) la selección de todos (*) los elementos de la tabla (SELECT * FROM LOGS). Usando un bucle for, el script recorre los elementos seleccionados (results) línea a línea (rows). Conserva los resultados para la visualización del contenido de la columna log (results.rows.item(i).log).

2. Método dinámico

Este método estático es muy rudimentario y los desarrolladores desean un sistema más elaborado que, a partir de un campo de un formulario, permita al usuario alimentar él mismo la base de datos.

Ejemplo

Al hacer clic en el botón **Añadir**, alimentamos la tabla con el contenido de una línea de texto del formulario.



Elements	Resources	Network	Sources	Timeline	Profiles	Audits	Console	X
▼ Folders	ID	lista	added_on					
► (ejemplo.html)	1	sas	Wed Apr 10 2013 13:00:08 GMT+0200 (Hora de verano romance)					
▼ Web SQL	2	Dato 1	Wed Apr 10 2013 13:02:21 GMT+0200 (Hora de verano romance)					
▼ sql	3	Dato 2	Wed Apr 10 2013 13:02:26 GMT+0200 (Hora de verano romance)					
list	4	Dato 3	Wed Apr 10 2013 13:02:34 GMT+0200 (Hora de verano romance)					
IndexedDB	5	Dato 4	Wed Apr 10 2013 13:02:38 GMT+0200 (Hora de verano romance)					
► Local Storage	6	Dato 5	Wed Apr 10 2013 13:06:28 GMT+0200 (Hora de verano romance)					
▼ Session Storage								

El código

```

<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<script type="text/javascript">
var dbase= {};
dbase.db = null;
dbase.open = function() {
dbase.db = openDatabase("sql", "1.0", "Description", 5 * 1024 *
1024);
}
dbase.createTable = function() {
var db = dbase.db;
db.transaction(function(tx) {
tx.executeSql("CREATE TABLE IF NOT EXISTS lista(ID INTEGER PRIMARY
KEY ASC, lista TEXT, added_on DATETIME)", []);
});
}
dbase.addlista = function(listaText) {
var db = dbase.db;
db.transaction(function(tx) {
var addedOn = new Date();
tx.executeSql("INSERT INTO lista(lista, added_on) VALUES (?,?)",
[listaText, addedOn],
dbase.onSuccess,
dbase.onError);
});
}
dbase.onError = function(tx, e) {
alert("Error: " + e.message);
}
dbase.onSuccess = function(tx, r) {
dbase.getAlllistItems(loadlistaItems);
}
dbase.getAlllistItems = function(renderFunc) {
var db = dbase.db;
db.transaction(function(tx) {
tx.executeSql("SELECT * FROM lista", [], renderFunc,
dbase.onError);
});
}
function loadlistaItems(tx, rs) {
var rowOutput = "";
var listaItems = document.getElementById("listaItems");
for (var i=0; i < rs.rows.length; i++) {
rowOutput += renderlista(rs.rows.item(i));
}
listaItems.innerHTML = rowOutput;
}
function renderlista(row) {

```

```

return "<li>" + row.lista + " </li>";
}
function init() {
dbase.open();
dbase.createTable();
dbase.getAlllistaItems(loadlistaItems);
}
function addlista() {
var lista = document.getElementById("lista");
dbase.addlista(lista.value);
lista.value = "";
}
</script>
</head>
<body onload="init();">
<h2>SQLite Database</h2>
<form type="post" onsubmit="addlista(); return false;">
<input type="text" id="lista" placeholder="Escriba sus datos"
size="20">
<input type="submit" value="Añadir">
</form>
<ul id="listaItems">
</ul>
</body>
</html>

```

Comentario

```

dbase.open = function() {
dbase.db = openDatabase("sql", "1.0", "Description", 5 * 1024 *
1024);
}

```

Creación de la base de datos sql.

```

dbase.createTable = function() {
var db = dbase.db;
db.transaction(function(tx) {
tx.executeSql("CREATE TABLE IF NOT EXISTS lista(ID INTEGER PRIMARY
KEY ASC, lista TEXT, added_on DATETIME)", []);
});
}

```

Creación de la tabla lista (CREATE TABLE IF NOT EXISTS lista). Esta tabla tiene tres columnas: la clave primaria, que es un entero (PRIMARY ID INTEGER KEY ASC), la columna lista, que es de tipo texto (lista TEXT) y la columna added_on, que será de tipo fecha y hora (added_onDATETIME).

```

dbase.addlista = function(listaText) {
var db = dbase.db;
db.transaction(function(tx){
var addedOn = new Date();
tx.executeSql("INSERT INTO lista(lista, added_on) VALUES (?,?)",
[listaText, addedOn],
dbase.onSuccess,
dbase.onError);
});
}

```

La transacción solicita a SQLite la inserción en la tabla lista (INSERT INTO lista), compuesta de las columnas lista y added_on (ver la creación de la tabla más arriba), de los valores todavía desconocidos en este momento (VALUES (?,?)). Estos valores serán proporcionados como argumentos de las funciones listaText y addedOn, definidos por la variable var addedOn =

```
new Date().  
  
dbase.onError = function(tx, e) {  
    alert("Error: " + e.message);  
}
```

La función de callback genérica de gestión de errores (`onError`) comunica el tipo de error en un cuadro de diálogo.

```
dbase.onSuccess = function(tx, r) {  
    dbase.getAlllistaItems(loadlistaItems);  
}
```

La función de callback en caso de éxito de la transacción (`onSuccess`) muestra los elementos de la tabla (ver más arriba).

```
dbase.getAlllistaItems = function(renderFunc) {  
    var db = dbase.db;  
    db.transaction(function(tx) {  
        tx.executeSql("SELECT * FROM lista", [], renderFunc,  
        dbase.onError);  
    });  
}
```

La función `getAlllistaItems` selecciona todos los elementos de la tabla `lista` (`SELECT * FROM lista`).

```
function loadlistaItems(tx, rs) {  
    var rowOutput = "";  
    var listaItems = document.getElementById("listaItems");  
    for (var i=0; i < rs.rows.length; i++) {  
        rowOutput += renderlista(rs.rows.item(i));  
    }  
    listaItems.innerHTML = rowOutput;  
}
```

Para todos los elementos de la tabla seleccionados, se ejecuta un bucle `for` que los comprueba registro a registro (`rows`) con objeto de retirar los datos `rs.rows.item(i)`. Se llama a la pequeña función `renderlista` (ver más adelante) para generar un ítem de lista con el dato `row.lista`. Todo está en la variable `rowOutput`, que se insertará en la etiqueta `` del cuerpo del documento identificado por `listaItems`.

```
function renderlista(row) {  
    return "<li>" + row.lista + " </li>";  
}
```

La función `renderlista(row)` inserta solo un ítem de lista (`row.lista`).

Eliminar datos

Para eliminar datos de un registro de la tabla:

```
"DELETE FROM nombre_de_la_tabla WHERE ID=x"
```

Es decir, eliminar de la tabla el registro correspondiente a la clave primaria x.

Ejemplo

Para cada registro, añadimos un enlace que permitirá eliminar el registro en cuestión.



Después de la eliminación:



ID	lista	added_on
2	Dato 1	Wed Apr 10 2013 13:02:21 GMT+0200 (Hora de verano romance)
3	Dato 2	Wed Apr 10 2013 13:02:26 GMT+0200 (Hora de verano romance)
4	Dato 3	Wed Apr 10 2013 13:02:34 GMT+0200 (Hora de verano romance)
5	Dato 4	Wed Apr 10 2013 13:02:38 GMT+0200 (Hora de verano romance)

El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style>
a { color: black;
    text-decoration: none}
</style>
<script type="text/javascript">
var dbase= {};
dbase.db = null;
dbase.open = function() {
dbase.db = openDatabase("sql", "1.0", "Description", 5 * 1024 *
1024);
}
dbase.createTable = function() {
var db = dbase.db;
db.transaction(function(tx) {
tx.executeSql("CREATE TABLE IF NOT EXISTS lista(ID INTEGER PRIMARY
KEY ASC, lista TEXT, added_on DATETIME)", []);
});
}
dbase.addlista = function(listaText) {
var db = dbase.db;
db.transaction(function(tx){
var addedOn = new Date();
tx.executeSql("INSERT INTO lista(lista, added_on) VALUES (?,?)",
[listaText, addedOn],
dbase.onSuccess,
dbase.onError);
});
}
dbase.onError = function(tx, e) {
alert("Error: " + e.message);
}
dbase.onSuccess = function(tx, r) {
dbase.getAlllistaItems(loadlistaItems);
}
dbase.getAlllistaItems = function(renderFunc) {
var db = dbase.db;
db.transaction(function(tx) {
tx.executeSql("SELECT * FROM lista", [], renderFunc,
dbase.onError);
});
}
dbase.deletelista = function(id) {
```

```

var db = dbase.db;
db.transaction(function(tx) {
tx.executeSql("DELETE FROM lista WHERE ID=?", [id],
dbase.onSuccess,
dbase.onError);
});
}
function loadlistaItems(tx, rs) {
var rowOutput = "";
var listaItems = document.getElementById("listaItems");
for (var i=0; i < rs.rows.length; i++) {
rowOutput += renderlista(rs.rows.item(i));
}
listaItems.innerHTML = rowOutput;
}
function renderlista(row) {
return "<li>" + row.lista + " [<a href='javascript:void(0);' onclick='dbase.deletelista(\"" + row.ID +"\");'>Eliminar</a>]</li>";
}
function init() {
dbase.open();
dbase.createTable();
dbase.getAlllistaItems(loadlistaItems);
}
function addlista() {
var lista = document.getElementById("lista");
dbase.addlista(lista.value);
lista.value = "";
}
</script>
</head>
<body onload="init();">
<h2> SQLite Database </h2>
<form type="post" onsubmit="addlista(); return false;">
<input type="text" id="lista" placeholder="Escriba sus datos" size="20">
<input type="submit" value="Añadir">
</form>
<ul id="listaItems">
</ul>
</body>
</html>

```

Comentario

```

function renderlista(row) {
return "<li>" + row.lista + " [<a href='javascript:void(0);' onclick='dbase.deletelista(\"" + row.ID +"\");'>Eliminar</a>]</li>";
}

```

A la función `renderlista`, que recordemos genera un ítem de lista no ordenada, le hemos añadido el enlace **Eliminar**. Al hacer clic en este enlace, se llama a la función `deletelista`, que recibe como argumento la clave primaria del registro del dato (`row.ID`).

```

dbase.deletelista = function(id) {
var db = dbase.db;
db.transaction(function(tx) {
tx.executeSql("DELETE FROM lista WHERE ID=?", [id],
dbase.onSuccess,
dbase.onError);
});
}

```

La función `deletelist`a elimina de la tabla (DELETE FROM lista WHERE ID=?", [id]) el registro correspondiente a la clave primaria que se recibe como argumento (`id`).

Modificar datos

El SQL que se va a utilizar es:

```
"UPDATE nombre_de_la_tabla SET nombre_de_la_columna_1=?,
nombre_de_la_columna_2=? WHERE id = x", [nuevo_valor_de_la_columna_1,
nuevo_valor_de_la_columna_2]);
```

Es decir, actualizar (*update*) la tabla para las columnas indicadas, donde la clave primaria es x. Los nuevos valores se proporcionan entre corchetes.

Ejemplo

Modificamos los datos de la clave primaria 3.

La situación inicial es:

Los API JavaScript de HTL file:///C/Trabajo/Trat SQLite Database

Escriba sus datos Añadir

- Dato 1 [Eliminar]
- Dato 2 [Eliminar]
- Dato 3 [Eliminar]
- Dato 4 [Eliminar]
- Dato 5 [Eliminar]

Modification de la clave primaria 3

Nuevos datos Modificar

Elements Resources Network Sources Timeline Profiles Audits Console			
▼ Folders	ID	lista	added_on
► (chap5_7.htm)	2	Dato 1	Wed Apr 10 2013 13:02:21 GMT+0200 (Hora de verano romance)
▼ Web SQL	3	Dato 2	Wed Apr 10 2013 13:02:26 GMT+0200 (Hora de verano romance)
► sql	4	Dato 3	Wed Apr 10 2013 13:02:34 GMT+0200 (Hora de verano romance)
► lista	5	Dato 4	Wed Apr 10 2013 13:02:38 GMT+0200 (Hora de verano romance)
▼ IndexedDB	6	Dato 5	Wed Apr 10 2013 13:25:30 GMT+0200 (Hora de verano romance)
► Local Storage			
▼ Session Storage			
► Cookies			
► Application Cache			

Después de la modificación:



Screenshot of the Chrome DevTools Network tab showing a table of data from a Web SQL database named 'lista'. The table has columns 'ID', 'lista', and 'added_on'.

ID	lista	added_on
2	Dato 1	Wed Apr 10 2013 13:02:21 GMT+0200 (Hora de verano romance)
3	Dato 2	Wed Apr 10 2013 13:31:59 GMT+0200 (Hora de verano romance)
4	-----	Wed Apr 10 2013 13:02:34 GMT+0200 (Hora de verano romance)
5	Dato 4	Wed Apr 10 2013 13:02:38 GMT+0200 (Hora de verano romance)
6	Dato 5	Wed Apr 10 2013 13:25:30 GMT+0200 (Hora de verano romance)

El código

```

<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style>
a { color: black;
    text-decoration: none}
</style>
<script type="text/javascript">
var dbase= {};
dbase.db = null;
dbase.open = function() {
dbase.db = openDatabase("sql", "1.0", "Description", 5 * 1024 *
1024);
}
dbase.createTable = function() {
var db = dbase.db;
db.transaction(function(tx) {
tx.executeSql("CREATE TABLE IF NOT EXISTS lista(ID INTEGER PRIMARY
KEY ASC, lista TEXT, added_on DATETIME)", []);
});
}
dbase.addlista = function(listaText) {
var db = dbase.db;
db.transaction(function(tx){
var addedOn = new Date();
tx.executeSql("INSERT INTO lista(lista, added_on) VALUES (?,?)",
[listaText, addedOn],
dbase.onSuccess,
dbase.onError);
});
}
dbase.onError = function(tx, e) {
alert("Error: " + e.message);
}
dbase.onSuccess = function(tx, r) {
dbase.getAlllistaItems(loadlistaItems);
}
dbase.getAlllistaItems = function(renderFunc) {
var db = dbase.db;
db.transaction(function(tx) {
tx.executeSql("SELECT * FROM lista", [], renderFunc,
dbase.onError);
});
}

```

```

}

dbase.deletelista = function(id) {
var db = dbase.db;
db.transaction(function(tx){
tx.executeSql("DELETE FROM lista WHERE ID=?", [id],
dbase.onSuccess,
dbase.onError);
});
}

dbase.update = function () {
var db = dbase.db;
var mod = document.getElementById("listabis").value
var addedOn = new Date();
db.transaction(function(tx) {
tx.executeSql("UPDATE lista SET lista=? , added_on=? WHERE id = 3",
[mod, addedOn],
dbase.onSuccess,
dbase.onError);
});
}

function loadlistaItems(tx, rs) {
var rowOutput = "";
var listaItems = document.getElementById("listaItems");
for (var i=0; i < rs.rows.length; i++) {
rowOutput += renderlista(rs.rows.item(i));
}
listaItems.innerHTML = rowOutput;
}

function renderlista(row) {
return "<li>" + row.lista + " [<a href='javascript:void(0);' onclick='dbase.deletelista(\"" + row.ID +");'>Eliminar</a>]
</li>";
}

function init() {
dbase.open();
dbase.createTable();
dbase.getAlllistaItems(loadlistaItems);
}

function addlista() {
var lista = document.getElementById("lista");
dbase.addlista(lista.value);
lista.value = "";
}

</script>
</head>
<body onload="init();">
<h2> SQLite Database </h2>
<form type="post" onsubmit="addlista(); return false;">
<input type="text" id="lista" placeholder="Escriba sus datos"
size="20">
<input type="submit" value="Añadir">
</form>
<ul id="listaItems">
</ul>
-----<br>
Modificación de la clave primaria 3<br>
<input type="text" id="listabis" placeholder="Nuevos datos"
size="20">
<button onclick="dbase.update()">Modificar</button>
</body>
</html>

```

Comentario

```
<button onclick="dbase.update()">Modificar</button>
```

Al hacer clic en el botón **Modificar** en el cuerpo del documento, se llama a la función dbase.update().

```
dbase.update = function () {
var db = dbase.db;
var mod = document.getElementById("listabis").value
var addedOn = new Date();
db.transaction(function(tx) {
tx.executeSql("UPDATE lista SET lista=?, added_on=? WHERE id = 3",
[mod,addedOn],
dbase.onSuccess,
dbase.onError);
});
}
```

La transacción asociada a la función dbase.update actualiza (UPDATE lista) las variables lista=? y added_on=?, que tienen como clave primaria el valor 3 (WHERE id = 3") con los nuevos valores que se dan en mod y addedOn.

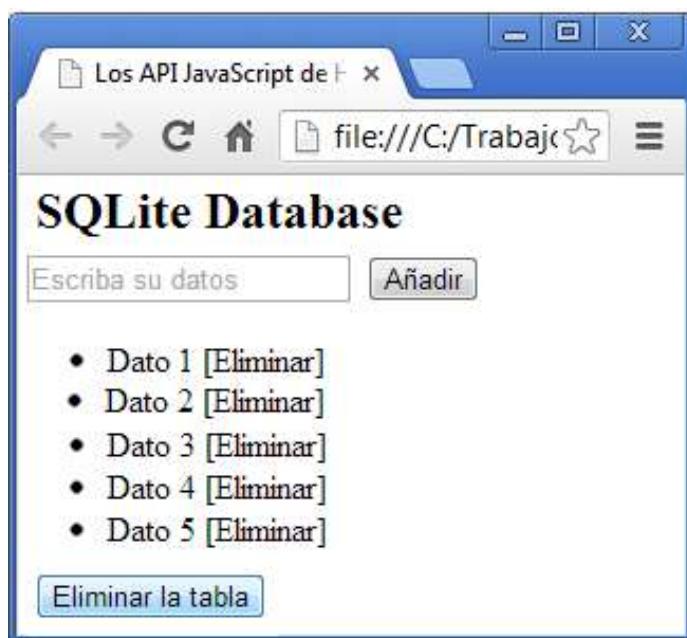
Eliminar una tabla de datos

La sintaxis SQL es simplemente:

```
DROP TABLE nombre_de_la_tabla
```

Ejemplo

Eliminamos la tabla lista.



Después de eliminar la tabla:





El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style>
a { color: black;
    text-decoration: none}
</style>
<script type="text/javascript">
var dbase= {};
dbase.db = null;
dbase.open = function() {
dbase.db = openDatabase("sql", "1.0", "Description", 5 * 1024 *
1024);
}
dbase.createTable = function() {
var db = dbase.db;
db.transaction(function(tx) {
tx.executeSql("CREATE TABLE IF NOT EXISTS lista(ID INTEGER PRIMARY
KEY ASC, lista TEXT, added_on DATETIME)", []);
});
}
dbase.addlista = function(listaText) {
var db = dbase.db;
db.transaction(function(tx){
var addedOn = new Date();
tx.executeSql("INSERT INTO lista(lista, added_on) VALUES (?,?)",
[listaText, addedOn],
dbase.onSuccess,
dbase.onError);
});
}
dbase.onError = function(tx, e) {
alert("Error: " + e.message);
}
dbase.onSuccess = function(tx, r) {
dbase.getAlllistaItems(loadlistaItems);
}
dbase.getAlllistaItems = function(renderFunc) {
var db = dbase.db;
db.transaction(function(tx) {
tx.executeSql("SELECT * FROM lista", [], renderFunc,
```

```

dbase.onError);
});

}

dbase.deletelista = function(id) {
var db = dbase.db;
db.transaction(function(tx){
tx.executeSql("DELETE FROM lista WHERE ID=?", [id],
dbase.onSuccess,
dbase.onError);
});
}

dbase.deletetable= function() {
var db = dbase.db;
db.transaction(function (tx) {
tx.executeSql("DROP TABLE lista;", [], null, dbase.onError);
}
);

location.reload();
}

function loadlistaItems(tx, rs) {
var rowOutput = "";
var listaItems = document.getElementById("listaItems");
for (var i=0; i < rs.rows.length; i++) {
rowOutput += renderlista(rs.rows.item(i));
}
listaItems.innerHTML = rowOutput;
}

function renderlista(row) {
return "<li>" + row.lista + " [<a href='javascript:void(0);' onclick='dbase.deletelista(" + row.ID +");'>Eliminar</a>]" +
"</li>";
}

function init() {
dbase.open();
dbase.createTable();
dbase.getAlllistaItems(loadlistaItems);
}

function addlista() {
var lista = document.getElementById("lista");
dbase.addlista(lista.value);
lista.value = "";
}

</script>
</head>
<body onload="init();">
<h2> SQLite Database </h2>
<form type="post" onsubmit="addlista(); return false;">
<input type="text" id="lista" placeholder="Escriba sus datos" size="20">
<input type="submit" value="Añadir">
</form>
<ul id="listaItems">
</ul>
<button onclick="dbase.deletetable()">Eliminar la tabla
</button>
</body>
</html>

```

Comentario

<button onclick="dbase.deletetable()">Eliminar la tabla</button>

Al hacer clic en el botón **Eliminar la tabla** en el cuerpo del documento, se llama a la función `dbase.deletetable()`.

```
dbase.deletetable= function() {  
var db = dbase.db;  
db.transaction(function (tx) {  
tx.executeSql("DROP TABLE lista;", [], null, dbase.onError);  
}  
);  
location.reload();  
}
```

La llamada al SQL es sencilla. Es suficiente con solicitar la eliminación de la tabla en cuestión (DROP TABLE lista). La página se recarga (reload) para que desaparezca la matriz, porque ya no hay tabla.

 Eliminamos la tabla de datos. En SQLite, no es posible eliminar la base de datos.

Aplicación final

Vamos a sintetizar los puntos anteriores para una aplicación que tiene varios datos y funcionalidades. Supongamos una libreta de direcciones con el nombre del contacto y su dirección de correo electrónico.

La pantalla inicial (en Google Chrome) se presenta de la siguiente manera:

The screenshot shows a browser window with a title bar 'Los API JavaScript'. The main content area has a heading 'Mis contactos'. Below it is a form with fields for 'Nombre' and 'Mail', and a 'Guardar' button. At the bottom is a 'Vaciar la tabla' button.

Añadimos algunos contactos.

The screenshot shows a browser window with a title bar 'Los API JavaScript'. The main content area has a heading 'Mis contactos' and a form with fields for 'Nombre' and 'Mail', and a 'Guardar' button. Below the form is a table listing four contacts:

Maria Sánchez	marias@hotmail.com	Eliminar
Julián Dagusti	juliad@hotmail.com	Eliminar
Ángel María Sánchez	angelms@hotmail.com	Eliminar
Mateo Sánchez	mateos@hotmail.com	Eliminar

At the bottom is a 'Vaciar la tabla' button.

Developer Tools - file:///C:/Trabajo/Trabajos%20Traducci%C3%B3n/JavaScript-HTML/EI5...

Elements Resources Network Sources Timeline Profiles Audits Console

	id	name	item
▼ Folders	1	Maria Sánchez	marias@hotmail.com
► (chap5_9.htm)	2	Julián Dagusti	juliad@hotmail.com
▼ Web SQL	3	Ángel María Sánchez	angelms@hotmail.com
► libretaDirecciones	4	Mateo Sánchez	mateos@hotmail.com
contact			
► IndexedDB			
► Local Storage			
▼ Session Storage			
► Cookies			
► Application Cache			

Es posible eliminar un contacto.

Mis contactos

Nombre:

Mail:

Maria Sánchez	marias@hotmail.com	<input type="button" value="Eliminar"/>
Julián Dagusti	juliad@hotmail.com	<input type="button" value="Eliminar"/>
Ángel María Sánchez	angelms@hotmail.com	<input type="button" value="Eliminar"/>
Mateo Sánchez	mateos@hotmail.com	<input type="button" value="Eliminar"/>

Después de la eliminación:

Los API JavaScript Los API JavaScript file:///C:/Trabajo/Trabajos%20Traducidos

Mis contactos

Nombre:

Mail:

Guardar

Maria Sánchez	marias@hotmail.com	Eliminar
Julián Dagusti	juliad@hotmail.com	Eliminar
Mateo Sánchez	mateos@hotmail.com	Eliminar

Vaciar la tabla

The screenshot shows a web-based contact manager. The interface includes a header with two tabs labeled 'Los API JavaScript'. The main content area has a title 'Mis contactos'. It contains a form with fields for 'Nombre' and 'Mail', and a 'Guardar' button. Below the form is a table with three rows of contact information. Each row has a 'Eliminar' button. At the bottom is a 'Vaciar la tabla' button.

Y para terminar, el botón **Vaciar la tabla** elimina la tabla y sus datos.

Los API JavaScript Los API JavaScript file:///C:/Trabajo/Trabajos%20Traducidos

Mis contactos

Nombre:

Mail:

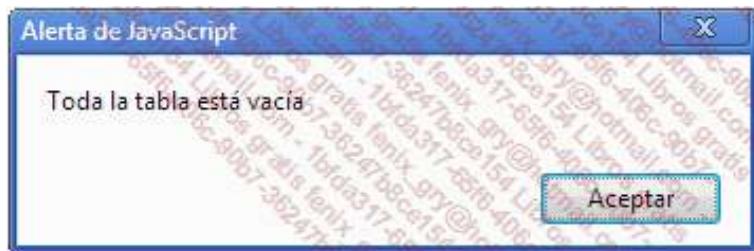
Guardar

Maria Sánchez	marias@hotmail.com	Eliminar
Julián Dagusti	juliad@hotmail.com	Eliminar
Mateo Sánchez	mateos@hotmail.com	Eliminar

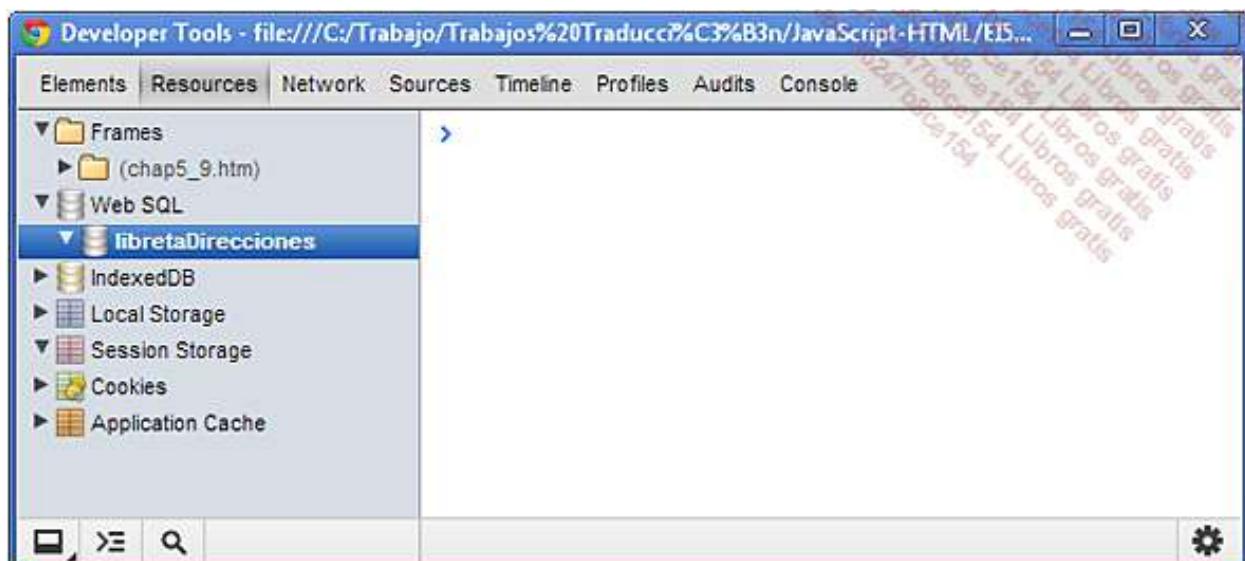
Vaciar la tabla

The screenshot shows the same contact manager application after the 'Vaciar la tabla' button was clicked. The table below the contact form is now empty, with only the column headers visible. The 'Eliminar' buttons remain next to each header cell.

Un cuadro de diálogo informa de que la tabla está vacía.



Lo que podemos verificar con Web Inspector.



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#box { border: 1px dotted black;
    padding: 5px;
    background-color: rgb(195,215,235);
    width: 250px; }
table { border: 1px solid gray;
    border-collapse: collapse; }
td { border: 1px solid gray;
    padding-left: 3px;
    padding-right: 3px; }
input { margin-left: 10px; }
</style>
<script type="text/javascript">
var db = window.openDatabase("LibretaDirecciones", "1.0",
"Description", 1024);
window.onload = function() {
if (!window.openDatabase) {
alert("Las bases de datos SQLite no están soportadas");
return;
}
db.transaction(function(tx) {
tx.executeSql("create table if not exists contact (
+ "id integer primary key,
+ "name text,
+ "item text
+ )",null,null, error);
```

```

showRecords();
});
function error(tx, error) {
alert("error");
}
document.querySelector("#save").onclick = function() {
db.transaction(function(tx) {
tx.executeSql(
"insert into contact (name, item) "
+ "values (?,?)", [document.querySelector("#name").value,
document.querySelector("#item").value], null, error
);
showRecords();
});
};
document.querySelector("#drop").onclick = function(ev) {
db.transaction(function(tx) {
tx.executeSql("drop table contact", [], null, error);
showRecords();
alert("Toda la tabla está vacía ");
html = "";
document.querySelector("#records").innerHTML = html;
});
}
document.querySelector("#records").onclick = function(ev) {
if (ev.target.className!="remove") return;
var recordID = ev.target.id;
db.transaction(function(tx) {
tx.executeSql("delete from contact where id=?", [recordID], null,
error);
showRecords();
});
}
function showRecords() {
html = "";
db.transaction(function(tx) {
tx.executeSql("select * from contact", [], function(tx, results)
{
for (var i=0; i<results.rows.length; i++) {
var record = results.rows.item(i);
html+="|  |  |  |
| --- | --- | --- |
| " + record.name + " | " + record.item + " | Eliminar |
";
}
document.querySelector("#records").innerHTML = html;
});
});
}
};

</script>
</head>
<body>
<h2>Mis contactos</h2>
<div id="box">
Nombre: <input id="name" value=""> <br>
Mail: &nbsp;<input id="item" value=""> <br>
<button id="save">Guardar</button>
</div>
<p>
<table id="records"></table>
</p><button id="drop">Vaciar la tabla</button>
</body>
</html>

```

Comentario

```
var db = window.openDatabase("LibretaDirecciones", "1.0",
"Description", 1024);
```

Creación de la base de datos LibretaDirecciones.

```
window.onload = function() {
if (!window.openDatabase) {
alert("Las bases de datos SQLite no están soportadas");
return;
}}
```

Verificación del soporte de las bases de datos SQLite.

```
db.transaction(function(tx) {
tx.executeSql("create table if not exists contact (
+ "id integer primary key,"
+ "name text,"
+ "item text"
+ ")"),null,null, error);
showRecords();
});
```

Creación de una tabla llamada contact (create table if not exists contact), que tiene una clave primaria (id), una columna para el nombre (name) y otra para la dirección del correo electrónico (item).

```
function error(tx, error) {
alert("error");
}
```

En caso de error, se muestra un cuadro de diálogo.

```
document.querySelector("#save").onclick = function() {
db.transaction(function(tx) {
tx.executeSql(
"insert into contact (name, item) "
+ "values (?,?)", [document.querySelector("#name").value,
document.querySelector("#item").value], null, error
);
showRecords();
});
```

La transacción va a insertar en la tabla contact (insert into contact(name, item)) los dos valores ("values (?,?)") correspondientes a los valores codificados en los campos del formulario, identificados por name e item(document.querySelector("#name").value).

```
document.querySelector("#drop").onclick = function(ev) {
db.transaction(function(tx) {
tx.executeSql("drop table contact",[], null, error);
showRecords();
alert("Toda la tabla está vacía ");
html = "";
document.querySelector("#records").innerHTML = html;
});
}
```

Al hacer clic en el botón drop (**Vaciar la tabla**), se produce una transacción SQL (executeSql) que va a eliminar la tabla contact (drop table contact). Un cuadro de diálogo confirma el tratamiento y la tabla de datos se elimina.

```
document.querySelector("#records").onclick = function(ev) {
```

```

if (ev.target.className!="remove") return;
var recordID = ev.target.id;
db.transaction(function(tx) {
tx.executeSql("delete from contact where id=?", [recordID], null,
error);
showRecords();
});
}

```

Estas líneas de código son para el botón **Eliminar**, que permite eliminar un contacto de la lista. En primer lugar hay que localizar la clave del registro (`id`), que se proporciona por el evento clic del botón (`ev.target.id`) y está contenida en la variable `recordID`. La consulta SQL `delete from contact where id=?`, `[recordID]`... elimina de la tabla `contact` el registro que corresponde a la clave primaria que hay en la variable `recordID`.

```

function showRecords() {
html = "";
db.transaction(function(tx) {
tx.executeSql("select * from contact", [], function(tx, results) {
for (var i=0; i<results.rows.length; i++) {
var record = results.rows.item(i);
html+="|  |  |  |
| --- | --- | --- |
| " + record.name + " | " + record.item + " |  |
";
}
document.querySelector("#records").innerHTML = html;
});
});
}
}

```

Para terminar, la llamada a la función `showrecords` que se encarga de imprimir todo el contenido de la tabla `contact`. En primer lugar, la consulta SQL `select * from contact` selecciona todos (*) los elementos de la tabla. Un bucle `for` recorre la tabla registro a registro (`rows`). Los elementos de cada registro (`results.rows.item(i)`) están contenidos en la variable `record`. Es suficiente con utilizar JavaScript clásico para construir registros en una tabla e insertar en ellos los valores contenidos en la variable `record`, es decir, `record.name`, `record.item` y el botón de eliminación.

Una vez codificados de esta manera los diferentes registros, el resultado final se incluye en la tabla identificada por `records`.

El futuro de Web SQL Database

El futuro de estas bases de datos a partir de SQLite, eficaces y fáciles de poner en marcha, todavía es incierto.

De hecho, el 18 noviembre de 2010, W3C anunció que esta especificación no evolucionaría más y que W3C no tenía la intención de mantenerla en el futuro.

Por tanto, estas bases de datos SQL están abocadas a desaparecer en los próximos años.

Las razones esgrimidas son las siguientes:

- SQLite es un elemento externo a Html5 y a JavaScript.
- SQLite no es una norma.
- Existe otra especificación, Indexed Database (ver el capítulo El API Indexed Database), que hace llamadas exclusivamente a JavaScript y cuyo código está alineado con los API JavaScript de Html5.
- La elección de SQLite la han hecho los fabricantes de los navegadores, en detrimento de la proposición inicial del API Indexed Database propuesto por W3C.
- Una presión por parte de Firefox, que ha desarrollado este API Indexed Database (hipótesis sugerida en algunos foros especializados).

En la práctica, estas bases de datos SQL Databases no van a desaparecer de la noche a la mañana debido a:

- La existencia de numerosas aplicaciones que ya las utilizan, particularmente los Smartphone. Falta ajustar algún detalle.
- La especificación de estas Indexed Databases todavía está en un estado poco avanzado, incluso experimental.
- El código de estas Indexed Databases es complicado de implementar.

La mejor opción para un desarrollador experimentado será reservar estas SQL Databases para las aplicaciones con una duración de vida limitada en el tiempo. Para las aplicaciones con una vida más prolongada, será necesario utilizar Indexed Databases en el futuro.

Google Chrome ha anunciado que abandona estas bases de datos SQL, pero solo en los próximos años.

Esto ilustra cómo Html5 todavía se halla en plena fase de diseño, lejos de estar terminado.

Presentación y objetivos

Si localStorage (cf. capítulo El almacenamiento de datos en local) es un avance notable en Html, ya que ha permitido almacenar algunos datos de manera permanente, el hecho de poder almacenar un número importante de datos estructurados en un sistema de base de datos es incluso más importante para el desarrollo de aplicaciones Web cada vez más elaboradas. Las bases de datos de Html5 van, sin ninguna duda, a modificar de manera radical las posibilidades que ofrece la Red.

Desde el abandono del API Web SQL Database (ver capítulo anterior), el API Indexed Database es la única norma oficial para la creación de bases de datos en Html5. Observe que esta especificación es relativamente joven. El último borrador (*draft*) se ha actualizado el 24 de mayo de 2012 y el primero se remonta al 5 de enero de 2010. Esto explica la corta experiencia de las primeras aplicaciones de este API en unos pocos navegadores.

El API Indexed Database retoma el concepto de pares clave/valor; la clave sirve de clave primaria, mientras que el valor aquí es un objeto JavaScript.

Sin sorpresas, la base de datos toma el nombre de database. Esta incluye los Object Stores (literalmente «almacenes de objetos») que son, de alguna manera, el equivalente a las tablas en SQL. Cada database puede contener varios Object Stores. A su vez, un Object Store contiene pares clave/valor que se describen más adelante.

Algunas precisiones:

- La base de datos se almacena de manera permanente en el navegador.
- Cuando se carga en el navegador, se puede consultar en línea o en modo desconectado (*offline*).
- Solo se gestiona usando JavaScript nativo y no hace llamadas a ninguna tecnología externa.
- De esta manera, todo está en el lado cliente. La consulta de la base de datos no necesita, por tanto, ninguna llamada al servidor. Esto implica una ganancia y mejora en la rapidez y fluidez.
- El proceso de este API es asíncrono. Este contribuye a la usabilidad de las aplicaciones, ya que las consultas se realizan sin afectar o ralentizar el proceso normal de la aplicación. Hay una versión síncrona de este API, pero está reservada a las Workers (capítulo El JavaScript en segundo plano).
- El API Indexed Database es una base de datos no SQL, como Mongo DB o CouchDB.
- Este API no es una base de datos relacional, sino orientada a objetos.
- No hay esquema fijo de registros y columnas. De manera esquemática, las claves representan los registros, y los objetos de valor, las diferentes columnas.
- El API Indexed Database es transaccional. Este impide que otros elementos de código JavaScript operen en la base de datos.
- Hace un uso extensivo de scripts. Las consultas devuelven éxito o error del evento DOM sobre el que operan.

Esto va a producir un código JavaScript más complejo. Lo descubrirá rápidamente cuando se traten los primeros ejemplos.

A nivel de seguridad, a la base de datos solo se podrá acceder por el dominio que ha creado la aplicación. Es decir, la misma política que para localStorage (cf. capítulo El almacenamiento de datos en local).

Firefox le solicita autorización para almacenar los datos.



Los ejemplos hacen necesario que esté conectado.

Disponibilidad del API

El API Indexed Database solo está soportado por IE en su versión 10, en Firefox a partir de la versión 4 y en Google Chrome a partir de su versión 11.

Estos primeros motores no están exentos de errores. En la práctica, priorice en la medida de lo posible las versiones más recientes de Firefox (19) y Google Chrome (25).

El API Indexed Database ya no se soporta en Safari, Opera e Internet Explorer 9. Para este último, solo está disponible con Internet Explorer 10. Estaba previsto para Safari 6 y Opera 12 (junio de 2012), pero de momento no está disponible.

Mientras esperamos una especificación más estable, incluso definitiva, algunas versiones de los navegadores mencionados utilizan el objeto indexedDB con un prefijo: moz para Firefox y webkit para Chrome. Internet Explorer 10 utiliza el prefijo ms. Esta práctica no despistará a los lectores acostumbrados a las hojas CSS3, que ya utilizan el mismo sistema de prefijos.

Como este API es de momento poco compatible, resulta prudente comprobar la disponibilidad.

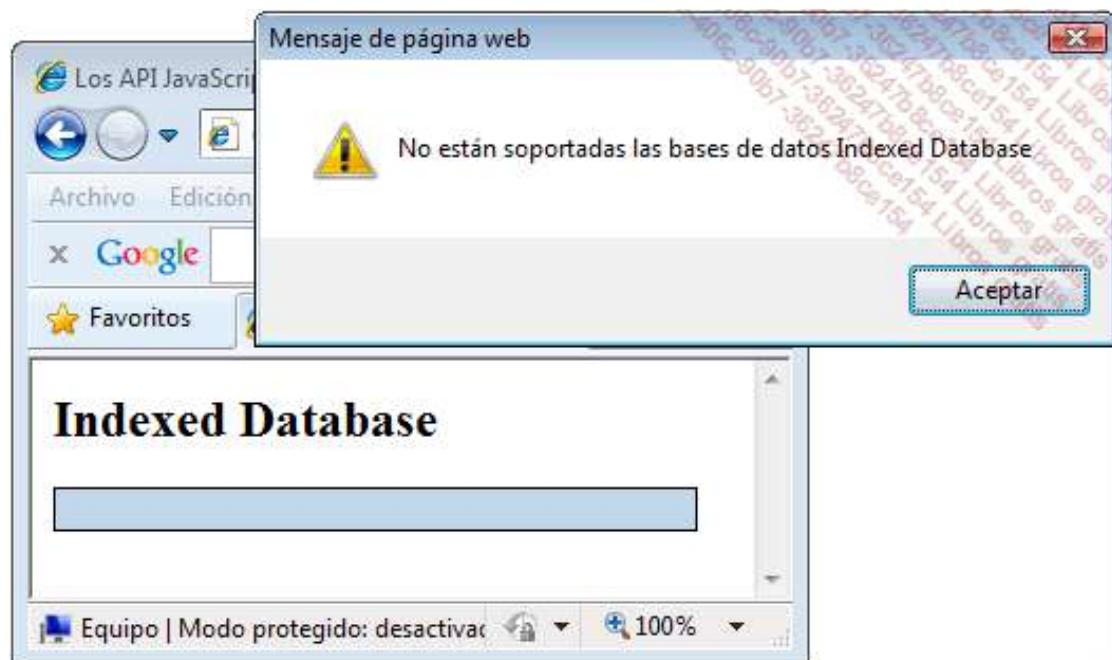
Ejemplo

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#box { width: 300px;
       border: 1px solid black;
       background-color: rgb(195,215,235);
       padding-left: 3px; }
</style>
<script type="text/javascript">
function init(){
var indexedDB = window.indexedDB || window.webkitIndexedDB ||
               window.mozIndexedDB;
if (indexedDB) {
msg = "El API Indexed Database está soportado";
document.querySelector('#box').innerHTML = msg;
}
else {
alert("No están soportadas las bases de datos Indexed Database")
}
}
</script>
</head>
<body onload="init();">
<h2>Indexed Database</h2>
<div id="box">&nbsp;</div>
</body>
</html>
```

En Firefox:



En Internet Explorer 9:



Crear una base de datos

La primera operación que hay que hacer es crear una nueva base de datos.

Es suficiente con utilizar el método `open()` del objeto `indexedDB`.

La sintaxis es:

```
indexedDB.open("nombre_de_la_base_de_datos", "descripción_de_la_base_de_datos")
```

El nombre de la base de datos es obligatorio, mientras que la descripción es opcional.

Si la base de datos no existe todavía, se crea. Si ya existe, simplemente se abre.

Esta operación se hace como una consulta.

Ejemplo

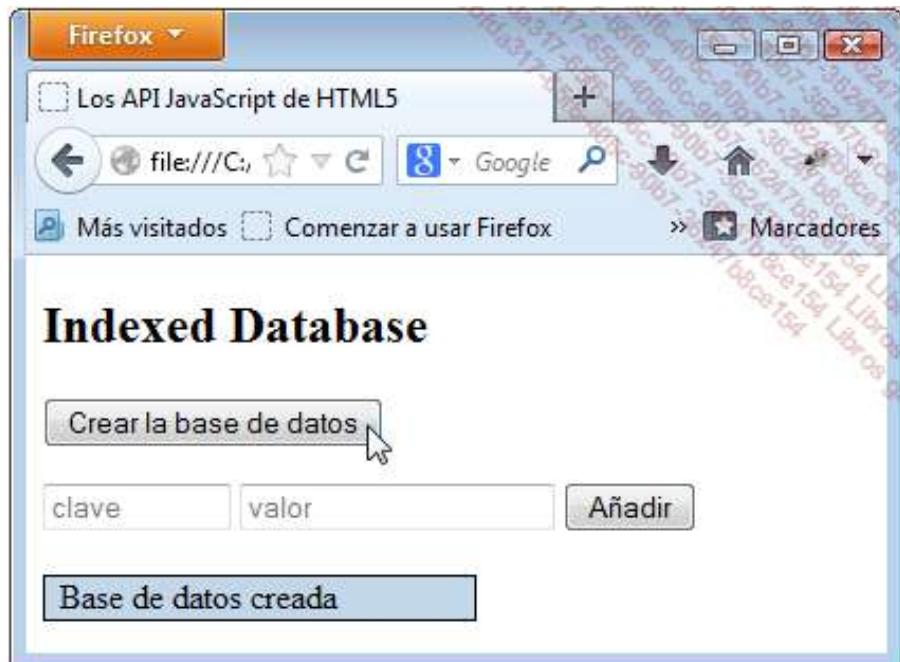
Veamos un pequeño script para la creación de una base de datos.

Recuerde que debe estar conectado para realizar este ejemplo.

La situación inicial es:



La creación de la base de datos:



Firebug es menos explícito que el Web Inspector de Safari y Chrome. Hay que bucear por el DOM para descubrir la presencia de la base de datos creada.

```
function() { constructor=IDBFactory, open=open(), IDBFactory = function() {} }  
deleteDatabase  
deleteForPrincipal  
open
```

El código

```
<!DOCTYPE html>  
<html lang="es">  
<head>  
<title>Los API JavaScript de Html5</title>  
<meta charset=utf-8>  
<style type="text/css">  
#idb-log { border: 1px solid black;  
            width: 200px;  
            background-color: #e6f2ff;  
            margin-top: 20px;  
            padding-left: 3px; }  
</style>  
<script type="text/javascript">  
function base(){  
    var idb;  
    var idbRequest;
```

```

var idbLog = document.getElementById('idb-log');
var idResultsWrapper =
document.getElementById('idb-results-wrapper');
if ('webkitIndexedDB' in window) {
window.indexedDB = window.webkitIndexedDB;
window.IDBTransacction = window.webkitIDBTransaction;
}
else if ('mozIndexedDB' in window) {
window.indexedDB = window.mozIndexedDB;
}
if (window.indexedDB) {
idbRequest = window.indexedDB.open("prueba", "test IndexedDB");
idbRequest.onerror = idbError;
idbRequest.addEventListener('success', function(e) {
idb = idbRequest.result
idbLog.innerHTML = "Base de datos creada";
}, false);
}
}
function idbError(e) {
idbLog.innerHTML += 'Error: ' +
e.message + ' (' + e.code + ')';
}
</script>
</head>
<body>
<h2>Indexed Database</h2>
<p>
<button onclick="base()">Crear la base de datos</button>
</p>
<input type="text" placeholder="clave" readonly id="idb-key"
size="10">
<input type="text" placeholder="valor" readonly id="idb-value"
size="20">
<button>Añadir</button>
<div id="idb-log">&nbsp;</div>
<div class="record-list" id="idb-results-wrapper"></div>
<p>
<div id='ourList'>
</p>
</body>
</html>

```

Comentario

```

var idb;
var idbRequest;

```

La variable idb define la base de datos local, y el objeto, la consulta utilizando la variable idbRequest.

```

if ('webkitIndexedDB' in window) {
window.indexedDB = window.webkitIndexedDB;
window.IDBTransacction = window.webkitIDBTransaction;
}
else if ('mozIndexedDB' in window) {
window.indexedDB = window.mozIndexedDB;
}

```

El siguiente código ilustra otra manera de probar la disponibilidad del API.

```

if (window.indexedDB) {
idbRequest = window.indexedDB.open("prueba", "test IndexedDB");
idbRequest.onerror = idbError;

```

```
idbRequest.addEventListener('success', function(e) {  
  idb = idbRequest.result  
  idbLog.innerHTML = "Base de datos creada";  
, false);  
})
```

Si el navegador soporta el API Indexed Database (if(window.indexedDB)), el script lanza una consulta (idbRequest) para crear (indexedDB.open) la base de datos prueba. En caso de error (idbRequest.onerror), se llama a la función idbError. Si el evento devuelve éxito para la consulta (addEventListener('success'...)), se muestra un mensaje (innerHTML) en la capa idbLog.

Crear un Object Store

Cualquier operación relativa a un Object Store debe comenzar por la definición de su versión. La sintaxis es:

```
db.setVersion()
```

Para crear o acceder a un Object Store, usaremos el método `createObjectStore()` aplicado a la base de datos.

```
db.createObjectStore("nombre_del_Object_Store")
```

Ejemplo

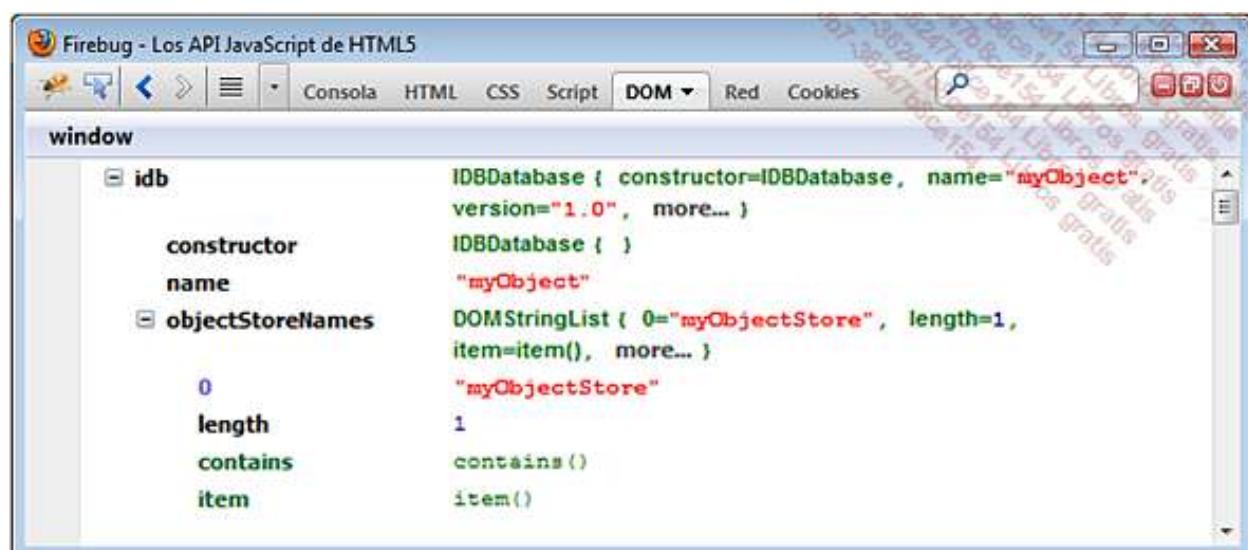
Vamos a crear un Object Store en nuestra base de datos.



Si el Object Store ya existe, se muestra un mensaje en la página.



Firebug nos dice un poco más.



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<style type="text/css">
#idb-log { border: 1px solid black;
            width: 200px;
            background-color: rgb(195,215,235);
            margin-top: 20px;
            padding-left: 3px; }
</style>
<script type="text/javascript">
var idb;
var idbRequest;
function base(){
var idbLog = document.getElementById('idb-log');
var idResultsWrapper =

```

```

document.getElementById('idb-results-wrapper');
if ('webkitIndexedDB' in window) {
window.indexedDB = window.webkitIndexedDB;
window.IDBTransacction = window.webkitIDBTransaction;
}
else if ('mozIndexedDB' in window) {
window.indexedDB = window.mozIndexedDB;
}
if (window.indexedDB) {
idbRequest = window.indexedDB.open("myObject", "Test IndexedDB");
idbRequest.onerror = idbError;
idbRequest.addEventListener('success', function(e) {
idb = idbRequest.result || e.result;
var request = idb.setVersion('1.0');
request.onerror = idbError;
request.onsuccess = function(e) {
if (!idb.objectStoreNames.contains('myObjectStore')) {
try {
var objectStore = idb.createObjectStore('myObjectStore', null);
idbLog.innerHTML = "Object Store creado ";
} catch (err) {
idbLog.innerHTML = '<p>' + err.toString() + '</p>';
}
}
else {
idbLog.innerHTML = " Object Store ya existe";
}
}
}, false);
}
}

function idbError(e) {
idbLog.innerHTML += '<p>Error: ' +
e.message + ' (' + e.code + ')</p>';
}

</script>
</head>
<body>
<h2>Indexed Database</h2>
<p>
<button onclick="base()">Crear Object Store</button>
</p>
<input type="text" placeholder="clave" readonly id="idb-key"
size="10">
<input type="text" placeholder="valor" readonly id="idb-value"
size="20">
<button>Añadir</button>
<div id="idb-log">&nbsp;</div>
<div class="record-list" id="idb-results-wrapper"></div>
<p>
<div id='ourList'></div>
</p>
</body>
</html>

```

Recordamos que este ejemplo está disponible en el página Información y es necesario estar conectado para ejecutarlo.

Comentario

```

if ('webkitIndexedDB' in window) {
window.indexedDB = window.webkitIndexedDB;
window.IDBTransacction = window.webkitIDBTransaction;
}
else if ('mozIndexedDB' in window) {

```

```
window.indexedDB = window.mozIndexedDB;  
}
```

Se muestra el aspecto experimental de este API, que, en función del navegador, utiliza los prefijos webkit o moz.

```
idbRequest.addEventListener('success', function(e) {  
  idb = idbRequest.result || e.result;
```

Firefox 4 utiliza idbRequest.result. A partir de Firefox 5 y Chrome, basta con e.result. Es necesario tener esto en cuenta.

```
var request = idb.setVersion('1.0');
```

Una consulta fija la versión del futuro Object Store a la versión 1.0.

```
request.onsuccess = function(e) {  
  if (!idb.objectStoreNames.contains('myObjectStore')) {  
    try {  
      var objectStore = idb.createObjectStore('myObjectStore', null);  
      idbLog.innerHTML = "Object Store creado ";  
    } catch (err) {  
      idbLog.innerHTML = '<p>' + err.toString() + '</p>';  
    }  
  }  
  else {  
    idbLog.innerHTML = " Object Store ya existe";  
  }  
}, false);  
}
```

Para una prueba condicional, el script comprueba si no existe, entre los Object Stores de la baseprueba, un Object Store cuyo nombre sea myObjectStore(!idb.objectStoreNames.contains('myObjectStore')). En caso negativo, se crea myObjectStore(idb.createObjectStore(...)) y se muestra un mensaje que confirma su creación. En caso afirmativo, muestra otro mensaje indicando que este Object Store ya existe.

Añadir, seleccionar y visualizar datos

Cualquier adición de datos se debe hacer utilizando una transacción. Este proceso asegura la base de datos frente a otros procesos JavaScript que podrían intentar modificar los mismos datos. Por ejemplo, podría ser el caso de una segunda ventana en el mismo navegador que fuera capaz de modificar simultáneamente el mismo dato.

Los datos se añaden al Object Store usando los métodos `store.put()` y `store.add()`. La diferencia entre ellos es que `put()` también puede actualizar los datos existentes. Intentar añadir un objeto con una clave inexistente fracasará.

Para localizar y visualizar los datos del Object Store, hay que crear un `IDBKeyRange`(ver más adelante) que, aplicado al método `store.opencursor()`, devuelva en caso de éxito un objeto cursor. Un cursor contendrá los datos del primer registro encontrado. La llamada al método `result.continue()` itera sobre el resto de los elementos.

El API Indexed Database incluye los objetos relacionados con las transacciones particulares. A continuación mostramos los más importantes:

<code>IDBDatabase</code>	Representa una conexión a una base de datos. Es la única manera de obtener una transacción en la base de datos.
<code>IDBObjectStore</code>	Representa un Object Store.
<code>IDBRequest</code>	Permite acceder a los resultados de las preguntas asíncronas, a las bases de datos y a los objetos de esta.
<code>IDBTransaction</code>	Representa una transacción. Es necesario indicar su alcance. <code>IDBTransaction.Read_Only</code> , en modo solo lectura. <code>IDBTransaction.Read_Write</code> , en modo lectura y escritura.
<code>IDBCursor</code>	Representa el objeto cursor que permite revisar los elementos del Object Store.
<code>IDB.objectStoreNames</code>	Permite acceder a los nombres de los diferentes Object Stores.
<code>IDB.deleteObjectStore</code>	Elimina un Object Store.
<code>IDB.setVersion</code>	Actualiza la versión del Object Store.
<code>IDBKeyRange.lowerBound</code>	Devuelve las claves inferiores a una clave dada.
<code>IDBKeyRange.upperBound</code>	Devuelve las claves superiores a una clave dada.

Para obtener una lista más completa de los IDB, consulte los sitios Web:

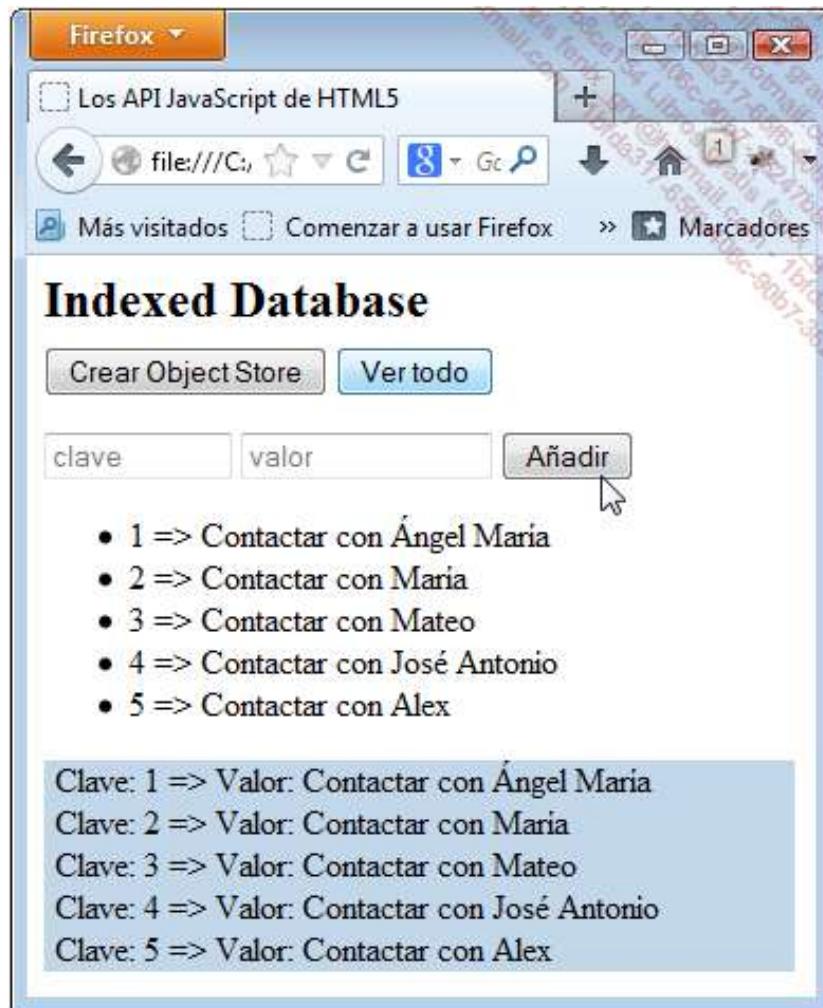
- https://developer.mozilla.org/en/IndexedDB/Basic_Concepts_Behind_IndexedDB
- <https://developer.mozilla.org/en/IndexedDB/>

Una transacción es un acceso a una base de datos para realizar una tarea. Una consulta es una operación asíncrona del estilo Ajax. En los ejemplos descubrirá que el código ejecuta una transacción hacia un Object Store, que consiste en una consulta asíncrona.

Ejemplo



El script también ofrece la posibilidad de visualizar la totalidad de la base de datos.



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#ourList { width: 280px;
            background-color: rgb(195,215,235);
            padding-left: 5px; }

</style>
</head>
<body>
<h2>Indexed Database</h2>
<p>
<button onclick="indexedDbUtil.idbCreate()">Crear Object Store</button>
<button onclick="indexedDbUtil.showAll()">Ver todo</button>
</p>
<input type="text" placeholder="clave" id="idb-key" size="10">
<input type="text" placeholder="valor" id="idb-value" size="15">
<button onclick="indexedDbUtil.idbSet()">Añadir</button>
<div id="idb-log"></div>
<div class="record-list" id="idb-results-wrapper"></div>
<p>
<div id='ourList'>
</div>
</p>
<script type="text/javascript">
var indexedDbUtil = (function() {
```

```
var idb_;
var idbRequest_;
var idbLog_ = document.getElementById('idb-log');
var idResultsWrapper_ =
document.getElementById('idb-results-wrapper');
if ('webkitIndexedDB' in window) {
window.indexedDB = window.webkitIndexedDB;
window.IDBTransacction = window.webkitIDBTransaction;
}
else if ('mozIndexedDB' in window) {
window.indexedDB = window.mozIndexedDB;
}
if (window.indexedDB) {
idbRequest_ = window.indexedDB.open("Test", "Test IndexedDB");
idbRequest_.onerror = idbError_;
idbRequest_.addEventListener('success', function(e) {
idb_ = idbRequest_.result || e.result;
idbShow_(e);
}, false);
}
function idbError_(e) {
idbLog_.innerHTML += '<p>Error: ' + e.message + ' (' + e.code + ')'
</p>';
}
function idbShow_(e) {
document.getElementById("idb-key").value="";
document.getElementById("idb-value").value="";
if (!idb_.objectStoreNames.contains('myObjectStore')) {
idbLog_.innerHTML = "<p>Object Store todavía no se ha creado</p>";
return;
}
var msgBoard = [];
var transaction = idb_.transaction(idb_.objectStoreNames,
IDBTransaction.READ_ONLY);
var request =
transaction.objectStore("myObjectStore").openCursor();
request.onsuccess = function(e) {
var cursor = request.result || e.result;
if (!cursor) {
idResultsWrapper_.innerHTML = '<ul class="record-list" id="idb-results">' + msgBoard.join('') + '</ul>';
return;
}
msgBoard.push('<li><span class="keyval" contenteditable
onblur="indexedDbUtil.updateKey(\',
cursor.key, \', this)">', cursor.key, '</span> ',
'=> <span class="keyval" contenteditable
onblur="indexedDbUtil.updateValue(\',
cursor.key, \', this)">', cursor.value, '</span>&nbsp; ',
'</li>');
cursor.continue();
}
request.onerror = idbError_;
}
function showAll_() {
document.getElementById("ourList").innerHTML = "";
var request = window.indexedDB.open("Test", "Test IndexedDB");
request.onsuccess = function(event) {
event.currentTarget.result.objectStoreNames("myObjectStore").open
Cursor();
var transaction = idb_.transaction(idb_.objectStoreNames,
IDBTransaction.READ_ONLY);
var request =
transaction.objectStore("myObjectStore").openCursor();
request.onsuccess = function(event) {
var cursor = request.result || event.result;

```

```
if (!cursor) {
return;
}
var element = document.createElement("div");
element.textContent = "clave: " + cursor.key + " => valor:
" + cursor.value;
document.getElementById("ourList").appendChild(element);
cursor.continue ();
};

};

}

function idbCreate_() {
if (!idb_) {
if (idbRequest_) {
idbRequest_.addEventListener('success', idb_.removeObjectStore,
false);
}
return;
}
var request = idb_.setVersion('the new version string');
request.onerror = idbError_;
request.onsuccess = function(e) {
if (!idb_.objectStoreNames.contains('myObjectStore')) {
try {
var objectStore = idb_.createObjectStore('myObjectStore', null);
idbLog_.innerHTML = "<p>Object Store creado </p>";
} catch (err) {
idbLog_.innerHTML = '<p>' + err.toString() + '</p>';
}
}
else {
idbLog_.innerHTML = '<p>Object Store ya existe</p>';
}
}
}

function idbSet_() {
if (!idb_) {
if (idbRequest_) {
idbRequest_.addEventListener('success', idb_.removeObjectStore,
false);
}
return;
}
if (!idb_.objectStoreNames.contains('myObjectStore')) {
idbLog_.innerHTML = "<p>Object Store no existe</p>";
return;
}
var objectStore = idb_.transaction(idb_.objectStoreNames,
IDBTransaction.READ_WRITE)
.objectStore("myObjectStore");
var request = objectStore.put(
document.getElementById('idb-value').value,
document.getElementById('idb-key').value);
request.onerror = idbError_;
request.onsuccess = idbShow_;
}
return {
idbSet: idbSet_,
idbCreate: idbCreate_,
showAll: showAll_
}
})();
</script>
</body>
</html>
```

Comentario

```
var indexedDbUtil = (function() {
```

La variable indexedDbUtil encapsula en una función las otras variables del script.

El script aborda lo que se trató en los puntos anteriores para la creación de la base de datos y del Object Store (idbCreate_()).

La función idbShow_(e) muestra los datos en forma de lista de opciones a medida que se crea en la capa identificada por idbresults-wrapper.

```
function idbShow_(e) {  
...  
var msgBoard = [];
```

La variable msgBoard contiene los elementos del mensaje que se va a visualizar.

```
var transaction = idb_.transaction(idb_.objectStoreNames,  
IDBTransaction.READ_ONLY);
```

Una transacción abre el Object Store en modo de solo lectura (READ_ONLY).

```
var request = transaction.objectStore("myObjectStore").openCursor();
```

Asociamos a esta transacción una consulta para recuperar el contenido del primer registro (openCursor()).

```
request.onsuccess = function(e) {  
var cursor = request.result || e.result;  
...  
}
```

En caso de éxito (onsuccess) de la consulta, el script genera el contenido de la lista de opciones. El contenido del registro se almacena en la variable cursor.

En la elaboración del mensaje (msgBoard.push), usamos cursor.key, que recupera el valor de la clave del dato, y cursor.value, que tiene el valor asociado.

```
cursor.continue();
```

Con cursor.continue(), se comprueba el resto de los datos del Object Store.

La función showAll_() muestra todos los datos creados en la capa azulada identificada por id='ourList'.

```
function showAll_() {  
document.getElementById("ourList").innerHTML = "";  
var request = window.indexedDB.open("Test", "Test IndexedDB");
```

La variable request abre (porque ya existe) la base de datos Test.

```
var transaction = idb_.transaction(idb_.objectStoreNames,  
IDBTransaction.READ_ONLY);  
var request =  
transaction.objectStore("myObjectStore").openCursor();  
request.onsuccess = function(event) {  
var cursor = request.result || event.result;
```

Siguiendo un esquema clásico (o casi...), la transacción abre el Object Store en modo de solo lectura (READ_ONLY). La consulta solicita recuperar los datos del primer registro (openCursor()). El resultado se almacena en la variable cursor.

```
if (!cursor) {  
    return;  
}
```

Si la variable `cursor` devuelve el valor `null`, la iteración por los datos no tiene lugar.

```
var element = document.createElement("div");  
element.textContent = "clave: " + cursor.key + " => valor:  
" + cursor.value;  
document.getElementById("ourList").appendChild(element);
```

Se crea la lista completa de los datos del Object Store en la capa gris del ejemplo. Los valores de `cursor.key` y de `cursor.value` se recuperan en la variable `element`. Esta se añade (`appendChild()`) a la capa `ourList` del cursor.

```
cursor.continue();
```

La comprobación de los otros registros continúa.

```
<button onclick="indexedDbUtil.idbSet()">Añadir</button>
```

En el cuerpo del documento, la función `idbSet_()` se asocia al botón **Añadir**.

```
function idbSet_() {  
    ...  
    var objectStore = idb_.transaction(idb_.objectStoreNames,  
        IDBTransaction.READ_WRITE).objectStore("myObjectStore");  
    var request = objectStore.put(  
        document.getElementById('idb-value').value,  
        document.getElementById('idb-key').value);  
    ...  
}
```

La variable `objectStore` lanza una transacción en modo lectura/escritura (`IDBTransaction.READ_WRITE`) sobre el Object Store `myObjectStore`. Para tratar los datos, el script hace una consulta (`var request`) que va a guardar, en la base de datos definida por la transacción (`objectStore.put`), el contenido (`value`) de los registros de texto identificados por `idb-value` e `idb-key`.

Eliminar datos

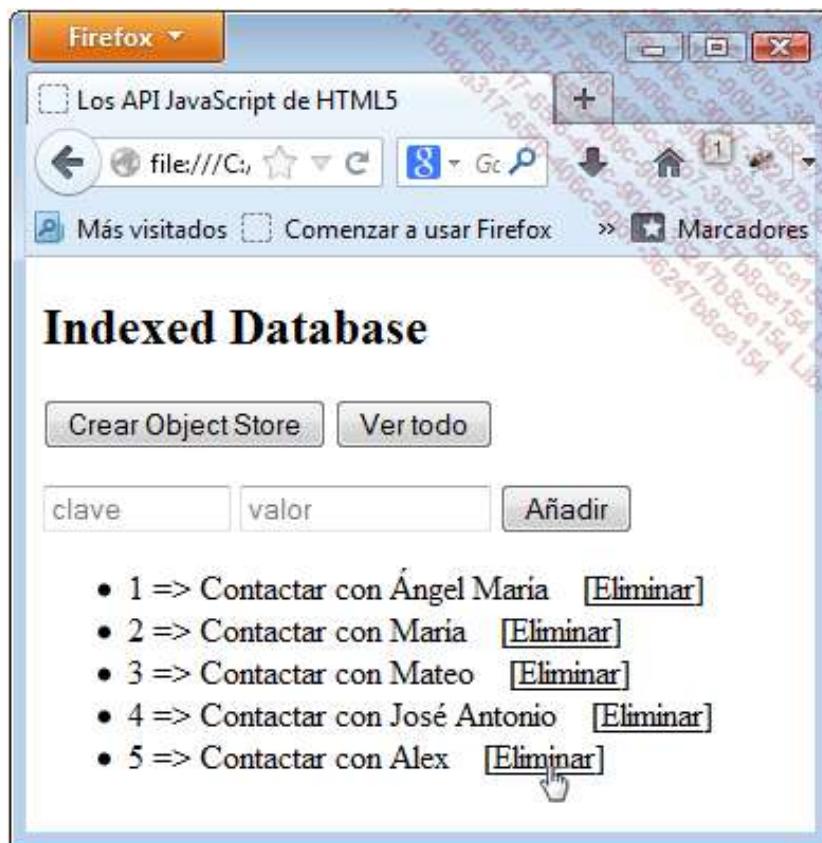
Para eliminar un dato del Object Store, es suficiente con utilizar la siguiente sintaxis:

```
remove(clave)
```

Firefox 4 usa `delete(clave)`.

Ejemplo

Vamos a eliminar un elemento del Object Store, en nuestro ejemplo una persona de contacto.



El resultado:



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#ourList { width: 280px;
            background-color: rgb(195,215,235);
            padding-left: 5px;}
a {cursor: pointer;
  text-decoration: none;
  color: black;}
</style>
</head>
<body>
<h2>Indexed Database</h2>
<p>
<button onclick="indexedDbUtil.idbCreate()">Crear Object Store
</button>
<button onclick="indexedDbUtil.showAll()">Ver todo</button>
</p>
<input type="text" placeholder="clave" id="idb-key" size="10">
<input type="text" placeholder="valor" id="idb-value" size="15">
<button onclick="indexedDbUtil.idbSet()">Añadir</button>
<div id="idb-log"></div>
<div class="record-list" id="idb-results-wrapper"></div>
<p>
<div id='ourList'>
</div>
</p>
<script type="text/javascript">
var indexedDbUtil = (function() {
var idb_;
var idbRequest_;
var idbLog_ = document.getElementById('idb-log');
```

```
var idResultsWrapper_ =  
document.getElementById('idb-results-wrapper');  
if ('webkitIndexedDB' in window) {  
window.indexedDB = window.webkitIndexedDB;  
window.IDBTransacction = window.webkitIDBTransaction;  
}  
else if ('mozIndexedDB' in window) {  
window.indexedDB = window.mozIndexedDB;  
}  
if (window.indexedDB) {  
idbRequest_ = window.indexedDB.open("Test", "Test IndexedDB");  
idbRequest_.onerror = idbError_;  
idbRequest_.addEventListener('success', function(e) {  
idb_ = idbRequest_.result || e.result;  
idbShow_(e);  
, false);  
}  
function idbError_(e) {  
idbLog_.innerHTML += '<p>Error: ' + e.message + ' (' + e.code +  
)</p>';  
}  
function idbShow_(e) {  
document.getElementById("idb-key").value="";  
document.getElementById("idb-value").value="";  
if (!idb_.objectStoreNames.contains('myObjectStore')) {  
idbLog_.innerHTML = "<p>Object Store todavía no se ha creado</p>";  
return;  
}  
var msgBoard = [];  
var transaction = idb_.transaction(idb_.objectStoreNames,  
IDBTransaction.READ_ONLY);  
var request =  
transaction.objectStore("myObjectStore").openCursor();  
request.onsuccess = function(e) {  
var cursor = request.result || e.result;  
if (!cursor) {  
idResultsWrapper_.innerHTML = '<ul class="record-list"  
id="idb-results">' + msgBoard.join('') + '</ul>';  
return;  
}  
msgBoard.push('<li><span class="keyval" contenteditable  
onblur="indexedDbUtil.updateKey(\'',  
cursor.key, '\', this)">', cursor.key, '</span> ',  
'=> <span class="keyval" contenteditable  
onblur="indexedDbUtil.updateValue(\'',  
cursor.key, '\', this)">', cursor.value, '</span>&nbsp; ',  
'<a href="javascript:void(0)"  
onclick="indexedDbUtil.deleteKey(\'',  
cursor.key, '\')">[Eliminar]</a></li>');  
cursor.continue();  
}  
request.onerror = idbError_;  
}  
function showAll_() {  
document.getElementById("ourList").innerHTML = "";  
var request = window.indexedDB.open("Test", "Test IndexedDB");  
request.onsuccess = function(event) {  
var transaction = idb_.transaction(idb_.objectStoreNames,  
IDBTransaction.READ_ONLY);  
var request = transaction.objectStore("myObjectStore").openCursor();  
request.onsuccess = function(event) {  
var cursor = request.result || event.result;  
if (!cursor) {  
return;  
}  
var element = document.createElement("div");
```

```
element.textContent = "clave: " + cursor.key + " => valor: " +
cursor.value;
document.getElementById("ourList").appendChild(element);
cursor.continue ();
};

};

}

function idbCreate_() {
if (!idb_) {
if (idbRequest_) {
idbRequest_.addEventListener('success', idb_.removeObjectStore,
false);
}
return;
}
var request = idb_.setVersion('1.0');
request.onerror = idbError_;
request.onsuccess = function(e) {
if (!idb_.objectStoreNames.contains('myObjectStore')) {
try {
var objectStore = idb_.createObjectStore('myObjectStore', null);
idbLog_.innerHTML = "<p>Object Store creado </p>";
} catch (err) {
idbLog_.innerHTML = '<p>' + err.toString() + '</p>';
}
}
else {
idbLog_.innerHTML = '<p>Object Store ya existe</p>';
}
}
}

function idbSet_() {
if (!idb_) {
if (idbRequest_) {
idbRequest_.addEventListener('success', idb_.removeObjectStore,
false);
}
return;
}
if (!idb_.objectStoreNames.contains('myObjectStore')) {
idbLog_.innerHTML = "<p>Object Store no existe</p>";
return;
}
var objectStore = idb_.transaction(idb_.objectStoreNames,
IDBTransaction.READ_WRITE)
.objectStore("myObjectStore");
var request = objectStore.put(
document.getElementById('idb-value').value,
document.getElementById('idb-key').value);
request.onerror = idbError_;
request.onsuccess = idbShow_;
}
function deleteKey_(key) {
var transaction = idb_.transaction(idb_.objectStoreNames,
IDBTransaction.READ_WRITE);
var objectStore = transaction.objectStore("myObjectStore");
if (objectStore.delete) {
var request = objectStore.delete(key);
}
else {
var request = objectStore.remove(key);
}
request.onerror = idbError_;
request.onsuccess = idbShow_;
}
return {

```

```
    idbSet: idbSet_,  
    idbCreate: idbCreate_,  
    deleteKey: deleteKey_,  
    showAll: showAll_  
}  
})();  
</script>  
</body>  
</html>
```

Comentario

```
<a href="javascript:void(0)" onclick="indexedDbUtil.deleteKey(''  
cursor.key, '\')">[Eliminar]</a>
```

En el cuerpo del documento, hemos añadido al enlace **Eliminar** la función `deleteKey().cursor.key` proporciona como argumento a esta función la clave primaria del elemento que se desea eliminar.

```
var objectStore = transaction.objectStore("myObjectStore");
```

El script se conecta a nuestro Object Store.

```
if (objectStore.delete) {  
var request = objectStore.delete(key);  
}  
else {  
var request = objectStore.remove(key);  
}
```

Una sentencia condicional usa tanto `delete(key)` algunas veces como `remove(key)` otras, para ser compatible con las diferentes versiones de Firefox.

Actualizar datos

Para actualizar el valor de un dato, es suficiente con usar el método `store.put()` que se ha visto en este capítulo, en la sección Añadir, seleccionar y visualizar datos para, de alguna manera, eliminar el valor existente.

Ejemplo

Modificamos un elemento del Object Store. En este caso, el valor asociado a la clave primaria 3.

Al inicio:

3	Miguel Ángel	Añadir
---	--------------	--------

- 1 => Contactar con Ángel María [Eliminar]
- 2 => Contactar con María [Eliminar]
- 3 => Contactar con Mateo [Eliminar]
- 4 => Contactar con José Antonio [Eliminar]
- 5 => Contactar con Alex [Eliminar]

Después de la modificación:



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#ourList { width: 280px;
            background-color: rgb(195,215,235);
            padding-left: 5px;}
a {cursor: pointer;
  text-decoration: none;
  color: black;}
</style>
</head>
<body>
<h2>Indexed Database</h2>
<p>
<button onclick="indexedDbUtil.idbCreate()">Crear Object Store
</button>
<button onclick="indexedDbUtil.showAll()">Ver todo</button>
</p>
<input type="text" placeholder="clave" id="idb-key" size="10">
<input type="text" placeholder="valor" id="idb-value" size="15">
<button onclick="indexedDbUtil.idbSet()">Añadir</button>
<div id="idb-log"></div>
<div class="record-list" id="idb-results-wrapper"></div>
<p>
<div id='ourList'>
</div>
</p>
<script type="text/javascript">
var indexedDbUtil = (function() {
var idb_;
var idbRequest_;
var idbLog_ = document.getElementById('idb-log');
var idResultsWrapper_ =
```

```
document.getElementById('idb-results-wrapper');
if ('webkitIndexedDB' in window) {
window.indexedDB = window.webkitIndexedDB;
window.IDBTransacction = window.webkitIDBTransaction;
}
else if ('mozIndexedDB' in window) {
window.indexedDB = window.mozIndexedDB;
}
if (window.indexedDB) {
idbRequest_ = window.indexedDB.open("Test", "Test IndexedDB");
idbRequest_.onerror = idbError_;
idbRequest_.addEventListener('success', function(e) {
idb_ = idbRequest_.result || e.result;
idbShow_(e);
}, false);
}
function idbError_(e) {
idbLog_.innerHTML += '<p>Error: ' + e.message + ' (' + e.code +
')</p>';
}
function idbShow_(e) {
document.getElementById("idb-key").value="";
document.getElementById("idb-value").value="";
if (!idb_.objectStoreNames.contains('myObjectStore')) {
idbLog_.innerHTML = "<p>Object Store todavía no se ha creado</p>";
return;
}
var msgBoard = [];
var transaction = idb_.transaction(idb_.objectStoreNames,
IDBTransaction.READ_ONLY);
var request =
transaction.objectStore("myObjectStore").openCursor();
request.onsuccess = function(e) {
var cursor = request.result || e.result;
if (!cursor) {
idResultsWrapper_.innerHTML = '<ul class="record-list" id="idb-results">' + msgBoard.join('') + '</ul>';
return;
}
msgBoard.push('<li><span class="keyval" contenteditable onblur="indexedDbUtil.updateKey(\'',
cursor.key, '\', this)">', cursor.key, '</span> ',
'=> <span class="keyval" contenteditable onblur="indexedDbUtil.updateValue(\'',
cursor.key, '\', this)">', cursor.value, '</span>&nbsp; ',
'<a href="javascript:void(0)" onclick="indexedDbUtil.deleteKey(\'',
cursor.key, '\')">[Eliminar]</a></li>');
cursor.continue();
}
request.onerror = idbError_;
}
function showAll_() {
document.getElementById("ourList").innerHTML = "";
var request = window.indexedDB.open("Test", "Test IndexedDB");
request.onsuccess = function(event) {
var transaction = idb_.transaction(idb_.objectStoreNames,
IDBTransaction.READ_ONLY);
var request = transaction.objectStore("myObjectStore").openCursor();
request.onsuccess = function(event) {
var cursor = request.result || event.result;
if (!cursor) {
return;
}
var element = document.createElement("div");
element.textContent = "clave: " + cursor.key + " => valor: " +
element.textContent;
idResultsWrapper_.innerHTML = '<ul class="record-list" id="idb-results">' + msgBoard.join('') + '</ul>';
}
}
}
```

```
cursor.value;
document.getElementById("ourList").appendChild(element);
cursor.continue ();
};

};

}

function idbCreate_() {
if (!idb_) {
if (idbRequest_) {
idbRequest_.addEventListener('success', idb_.removeObjectStore,
false);
}
return;
}
var request = idb_.setVersion('1.0');
request.onerror = idbError_;
request.onsuccess = function(e) {
if (!idb_.objectStoreNames.contains('myObjectStore')) {
try {
var objectStore = idb_.createObjectStore('myObjectStore', null);
idbLog_.innerHTML = "<p>Object Store creado </p>";
} catch (err) {
idbLog_.innerHTML = '<p>' + err.toString() + '</p>';
}
}
else {
idbLog_.innerHTML = '<p>Object Store ya existe</p>';
}
}
}

function idbSet_() {
if (!idb_) {
if (idbRequest_) {
idbRequest_.addEventListener('success', idb_.removeObjectStore,
false);
}
return;
}
if (!idb_.objectStoreNames.contains('myObjectStore')) {
idbLog_.innerHTML = "<p>Object Store no existe</p>";
return;
}
var objectStore = idb_.transaction(idb_.objectStoreNames,
IDBTransaction.READ_WRITE)
.objectStore("myObjectStore");
var request = objectStore.put(
document.getElementById('idb-value').value,
document.getElementById('idb-key').value);
request.onerror = idbError_;
request.onsuccess = idbShow_;
}
function deleteKey_(key) {
var transaction = idb_.transaction(idb_.objectStoreNames,
IDBTransaction.READ_WRITE);
var objectStore = transaction.objectStore("myObjectStore");
if (objectStore.delete) {
var request = objectStore.delete(key);
}
else {
var request = objectStore.remove(key);
}
request.onerror = idbError_;
request.onsuccess = idbShow_;
}
function updateValue_(key, element) {
var transaction = idb_.transaction(idb_.objectStoreNames,
```

```

IDBTransaction.READ_WRITE);
var objectStore = transaction.objectStore("myObjectStore");
var request = objectStore.put(element.textContent, key);
request.onerror = idbError_;
request.onsuccess = idbShow_;
}
function deleteKey_(key) {
var transaction = idb_.transaction(idb_.objectStoreNames,
IDBTransaction.READ_WRITE);
var objectStore = transaction.objectStore("myObjectStore");
if (objectStore.delete) {
var request = objectStore.delete(key);
}
else {
var request = objectStore.remove(key);
}
request.onerror = idbError_;
request.onsuccess = idbShow_;
}
return {
idbSet: idbSet_,
idbCreate: idbCreate_,
deleteKey: deleteKey_,
updateValue: updateValue_,
showAll: showAll_
}
})();
</script>
</body>
</html>

```

Comentario

```

function updateValue_(key, element) {
var transaction = idb_.transaction(idb_.objectStoreNames,
IDBTransaction.READ_WRITE);
var objectStore = transaction.objectStore("myObjectStore");
var request = objectStore.put(element.textContent, key);
...

```

Una transacción abre el Object Store en lectura/escritura (READ_WRITE). Este se almacena en la variable transaction. Una consulta guarda (objectStore.put) el nuevo elemento (element.textContent) en la pareja con la clave primaria (key) implicada.

Eliminar el Object Store

La eliminación del Object Store se hace con `IDB.deleteObjectStore`, al que se le proporciona el nombre del Object Store que se desea eliminar.

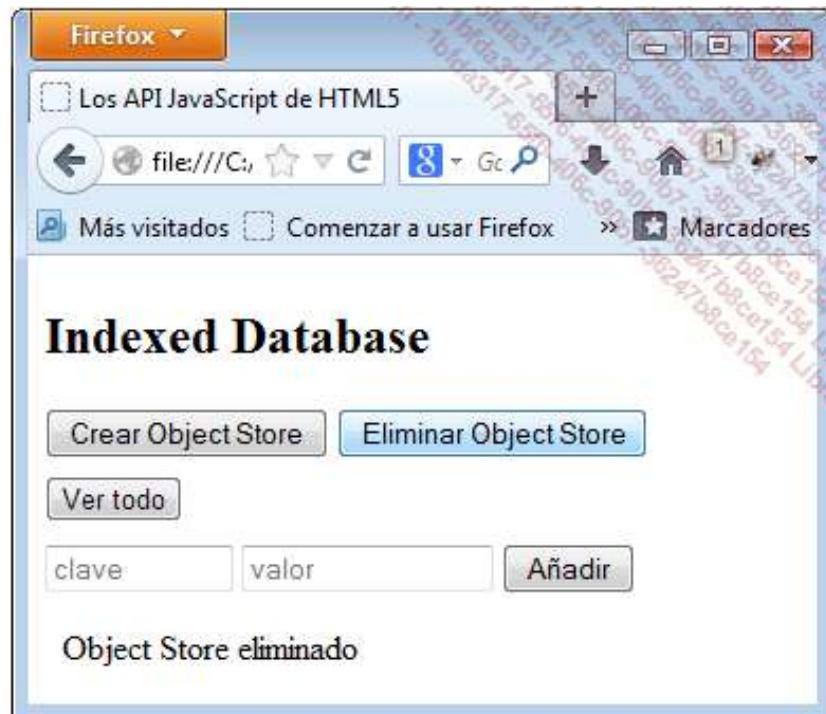
```
IDB.deleteObjectStore('nombre_del_Object_Store');
```

Observe que previamente se utilizó `IDB.removeObjectStore`.

Ejemplo

Eliminamos nuestro Object Store.





El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#ourList { width: 280px;
            background-color: rgb(195, 215, 235);
            padding-left: 5px; }
</style>
</head>
<body>
<h2>Indexed Database</h2>
<p>
<button onclick="indexedDbUtil.idbCreate()">Crear Object Store
</button>
<button onclick="indexedDbUtil.idbRemove()">Eliminar Object
Store</button>
<button onclick="indexedDbUtil.showAll()">Ver todo</button>
</p>
<input type="text" placeholder="clave" id="idb-key" size="10">
<input type="text" placeholder="valor" id="idb-value" size="15">
<button onclick="indexedDbUtil.idbSet()">Añadir</button>
<div id="idb-log"></div>
<div class="record-list" id="idb-results-wrapper"></div>
<p>
<div id='ourList'>
</div>
</p>
<script type="text/javascript">
var indexedDbUtil = (function() {
var idb_;
var idbRequest_;
var idbLog_ = document.getElementById('idb-log');
var idResultsWrapper_ =
document.getElementById('idb-results-wrapper');
if ('webkitIndexedDB' in window) {
window.indexedDB = window.webkitIndexedDB;
```

```
window.IDBTransacction = window.webkitIDBTransaction;
}
else if ('mozIndexedDB' in window) {
window.indexedDB = window.mozIndexedDB;
}
if (window.indexedDB) {
idbRequest_ = window.indexedDB.open("Test", "Test IndexedDB");
idbRequest_.onerror = idbError_;
idbRequest_.addEventListener('success', function(e) {
idb_ = idbRequest_.result || e.result;
idbShow_(e);
}, false);
}
function idbError_(e) {
idbLog_.innerHTML += '<p>Error: ' + e.message + ' (' + e.code +
')</p>';
}
function idbShow_(e) {
document.getElementById("idb-key").value="";
document.getElementById("idb-value").value="";
if (!idb_.objectStoreNames.contains('myObjectStore')) {
idbLog_.innerHTML = "<p>Object Store todavía no se ha creado</p>";
return;
}
var msgBoard = [];
var transaction = idb_.transaction(idb_.objectStoreNames,
IDBTransaction.READ_ONLY);
var request =
transaction.objectStore("myObjectStore").openCursor();
request.onsuccess = function(e) {
var cursor = request.result || e.result;
if (!cursor) {
idResultsWrapper_.innerHTML =
'<ul class="record-list">
id="idb-results">' + msgBoard.join('') + '</ul>';
return;
}
msgBoard.push('<li><span class="keyval" contenteditable
onblur="indexedDbUtil.updateKey(\',
cursor.key, '\', this)">', cursor.key, '</span> ',
'=> <span class="keyval" contenteditable
onblur="indexedDbUtil.updateValue(\',
cursor.key, '\', this)">', cursor.value, '</span>&nbsp; ',
'<a href="javascript:void(0)" onclick="indexedDbUtil.deleteKey(\',
cursor.key, '\')">[Eliminar]</a></li>');
cursor.continue();
}
request.onerror = idbError_;
}
function showAll_() {
document.getElementById("ourList").innerHTML = "";
var request = window.indexedDB.open("Test", "Test IndexedDB");
request.onsuccess = function(event) {
var transaction = idb_.transaction(idb_.objectStoreNames,
IDBTransaction.READ_ONLY);
var request =
transaction.objectStore("myObjectStore").openCursor();
request.onsuccess = function(event) {
var cursor = request.result || event.result;
if (!cursor) {
return;
}
var element = document.createElement("div");
element.textContent = "clave: " + cursor.key + " => valor: " +
cursor.value;
document.getElementById("ourList").appendChild(element);
}
}
}

```

```
cursor.continue();
};

};

}

function idbCreate_() {
if (!idb_) {
if (idbRequest_) {
idbRequest_.addEventListener('success', idb_.removeObjectStore,
false);
}
return;
}
var request = idb_.setVersion('1.0');
request.onerror = idbError_;
request.onsuccess = function(e) {
if (!idb_.objectStoreNames.contains('myObjectStore')) {
try {
var objectStore = idb_.createObjectStore('myObjectStore', null);
idbLog_.innerHTML = "<p>Object Store creado </p>";
} catch (err) {
idbLog_.innerHTML = '<p>' + err.toString() + '</p>';
}
}
else {
idbLog_.innerHTML = '<p>Object Store ya existe</p>';
}
}
}

function idbSet_() {
if (!idb_) {
if (idbRequest_) {
idbRequest_.addEventListener('success', idb_.removeObjectStore,
false);
}
return;
}
if (!idb_.objectStoreNames.contains('myObjectStore')) {
idbLog_.innerHTML = "<p>Object Store no existe</p>";
return;
}
var objectStore = idb_.transaction(idb_.objectStoreNames,
IDBTransaction.READ_WRITE)
.objectStore("myObjectStore");
var request = objectStore.put(
document.getElementById('idb-value').value,
document.getElementById('idb-key').value);
request.onerror = idbError_;
request.onsuccess = idbShow_;
}
function updateValue_(key, element) {
var transaction = idb_.transaction(idb_.objectStoreNames,
IDBTransaction.READ_WRITE);
var objectStore = transaction.objectStore("myObjectStore");
var request = objectStore.put(element.textContent, key);
request.onerror = idbError_;
request.onsuccess = idbShow_;
}
function deleteKey_(key) {
var transaction = idb_.transaction(idb_.objectStoreNames,
IDBTransaction.READ_WRITE);
var objectStore = transaction.objectStore("myObjectStore");
if (objectStore.delete) {
var request = objectStore.delete(key);
}
else {
var request = objectStore.remove(key);
}
}
}
```

```

}
request.onerror = idbError_;
request.onsuccess = idbShow_;
}
function idbRemove_() {
if (!idb_) {
if (idbRequest_) {
idbRequest_.addEventListener('success', idb_.removeObjectStore,
false);
}
return;
}
var request = idb_.setVersion("1.0");
request.onerror = idbError_;
request.onsuccess = function(e) {
if (idb_.objectStoreNames.contains('myObjectStore')) {
try {
if (idb_.deleteObjectStore) {
idb_.deleteObjectStore('myObjectStore');
}
else {
idb_.removeObjectStore('myObjectStore');
}
idResultsWrapper_.innerHTML = '';
idbLog_.innerHTML = "<p>Object Store eliminado.</p>";
} catch (err) {
idbLog_.innerHTML = '<p>' + err.toString() + '</p>';
}
}
else {
idbLog_.innerHTML = "<p>Object Store no existe</p>";
}
};
}
return {
idbSet: idbSet_,
idbCreate: idbCreate_,
idbRemove: idbRemove_,
updateValue: updateValue_,
deleteKey: deleteKey_,
showAll: showAll_
}
})();
</script>
</body>
</html>

```

Comentario

```
var request = idb_.setVersion("1.0");
```

Cualquier consulta para crear o eliminar un Object Store pasa por la definición de su versión.

```

request.onsuccess = function(e) {
if (idb_.objectStoreNames.contains('myObjectStore')) {
try {
if (idb_.deleteObjectStore) {
idb_.deleteObjectStore('myObjectStore');
}
else {
idb_.removeObjectStore('myObjectStore');
}
}

```

Se ejecuta una sentencia condicional para aplicar `deleteObjectStore` o `removeObjectStore`.

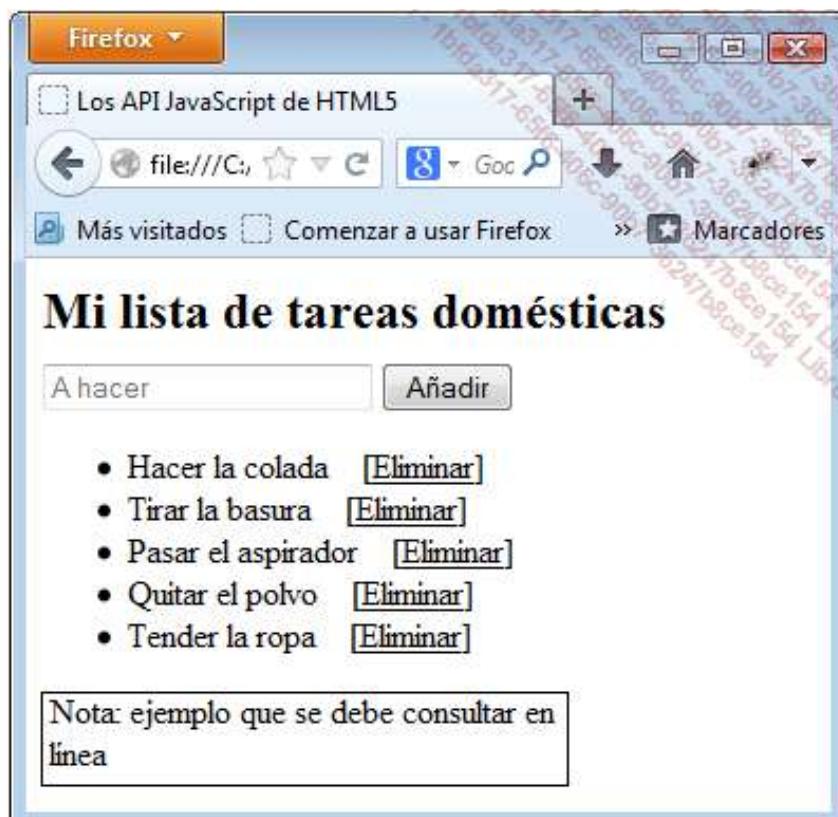
Aplicación final

Vamos a construir una lista de tareas domésticas para realizar, utilizando el sistema Indexed Database.

La situación inicial:



La lista completa:



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
```

```
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#note { border: 1px solid black;
         padding-left: 3px;
         width: 250px; }
a { cursor: pointer;
     color: black;
     text-decoration: none; }
</style>
<script type="text/javascript">
var html5 = {};
html5.indexedDB = {};
html5.indexedDB.db = null;
var indexedDB = window.indexedDB || window.webkitIndexedDB ||
               window.mozIndexedDB;
if ("webkitIndexedDB" in window) {
window.IDBTransacction = window.webkitIDBTransaction;
window.IDBKeyRange = window.webkitIDBKeyRange;
}
html5.indexedDB.onerror = function(e) {
console.log(e);
};
html5.indexedDB.open = function() {
var request = indexedDB.open("todo");
request.onsuccess = function(e) {
var v = "1.0";
html5.indexedDB.db = e.target.result;
var db = html5.indexedDB.db;
if (v!= db.version) {
var setVrequest = db.setVersion(v);
setVrequest.onerror = html5.indexedDB.onerror;
setVrequest.onsuccess = function(e) {
if(db.objectStoreNames.contains("todo")) {
db.deleteObjectStore("todo");
}
var almacen = db.createObjectStore("todo",
           {keyPath: "timeStamp"});
html5.indexedDB.getAllTodoItems();
};
}
else {
html5.indexedDB.getAllTodoItems();
}
};
request.onerror = html5.indexedDB.onerror;
}
html5.indexedDB.addTodo = function(todoText) {
var db = html5.indexedDB.db;
var trans = db.transaction(["todo"], IDBTransaction.READ_WRITE);
var almacen = trans.objectStore("todo");
var dato = { "text": todoText,
            "timeStamp": new Date().getTime()
};
var request = store.put(data);
request.onsuccess = function(e) {
html5.indexedDB.getAllTodoItems();
};
request.onerror = function(e) {
console.log("Error: ", e);
};
};
html5.indexedDB.deleteTodo = function(id) {
var db = html5.indexedDB.db;
var trans = db.transaction(["todo"], IDBTransaction.READ_WRITE);
var almacen = trans.objectStore("todo");

```

```

var request = store.delete(id);
request.onsuccess = function(e) {
html5.indexedDB.getAllTodoItems();
};

request.onerror = function(e) {
console.log("Error: ", e);
};

html5.indexedDB.getAllTodoItems = function() {
var todos = document.getElementById("todoItems");
todos.innerHTML = "";
var db = html5.indexedDB.db;
var trans = db.transaction(["todo"], IDBTransaction.READ_WRITE);
var almacen = trans.objectStore("todo");
var keyRange = IDBKeyRange.lowerBound(0);
var cursorRequest = store.openCursor(keyRange);
cursorRequest.onsuccess = function(e) {
var result = e.target.result;
if (!!result == false)
return;
renderTodo(result.value);
result.continue();
};
cursorRequest.onerror = html5.indexedDB.onerror;
};

function renderTodo(row) {
var todos = document.getElementById("todoItems");
var li = document.createElement("li");
var a = document.createElement("a");
var t = document.createTextNode(row.text);
a.addEventListener("click", function() {
html5.indexedDB.deleteTodo(row.timeStamp);
}, false);
a.textContent = " [Eliminar]";
li.appendChild(t);
li.appendChild(a);
todos.appendChild(li)
}
function addTodo() {
var todo = document.getElementById("todo");
html5.indexedDB.addTodo(todo.value);
todo.value = "";
}
function init() {
html5.indexedDB.open();
}
window.addEventListener("DOMContentLoaded", init, false);
</script>
</head>
<body>
<h2>Mi lista de tareas domésticas</h2>
<form>
<input type="text" id="todo" name="todo" placeholder="A hacer "
style="width: 150px;">
<input type="submit" value="Añadir" onclick="addTodo(); return
false;">
</form>
<ul id="todoItems"></ul>
<div id="note">Nota: ejemplo que se debe consultar en línea </div>
</body>
</html>

```

Comentario

```
var html5 = {};
```

```
html5.indexedDB = {};
```

Definición de la variable objeto html5 que va a encapsular la variable indexedDB que representa la base de datos.

```
html5.indexedDB.open = function() {
var request = indexedDB.open("todo");
```

Creación (indexedDB.open) de la base de datos todo.

```
request.onsuccess = function(e) {
var v = "1.0";
html5.indexedDB.db = e.target.result;
var db = html5.indexedDB.db;
if (v!= db.version) {
var setVrequest = db.setVersion(v);
setVrequest.onerror = html5.indexedDB.onerror;
setVrequest.onsuccess = function(e) {
if(db.objectStoreNames.contains("todo")) {
db.deleteObjectStore("todo");
}
var almacen = db.createObjectStore("todo",
{keyPath: "timeStamp"});
```

Creamos el Object Store. La variable v contiene el número de versión (1.0). Después de haber preguntado a la base de datos (html5.indexedDB.db = e.target.result) si la versión es diferente de 1.0, se asigna esta versión (db.setVersion(v)). Si el Object Store todo ya existe (if(db.objectStoreNames.contains("todo"))), el script lo elimina (db.deleteObjectStore("todo")). Entonces se crea un nuevo Object Store todo(db.createObjectStore("todo")). La clave primaria se construye con la fecha y la hora (*timestamp*).

```
html5.indexedDB.deleteAll = function(id) {
var db = html5.indexedDB.db;
var trans = db.transaction(["todo"], IDBTransaction.READ_WRITE);
var almacen = trans.objectStore("todo");
var request = store.delete(id);
request.onsuccess = function(e) {
html5.indexedDB.getAllTodoItems();
};
```

Para eliminar un dato del Object Store todo, se realiza una transacción en modo lectura/escritura (var trans) y el resultado se asigna a la variable store. Una consulta elimina el dato cuya clave primaria (id) es la que se proporciona como argumento (store.delete(id)).

```
html5.indexedDB.getAllTodoItems = function() {
var todo = document.getElementById("todoItems");
todos.innerHTML = "";
var db = html5.indexedDB.db;
var trans = db.transaction(["todo"], IDBTransaction.READ_WRITE);
var almacen = trans.objectStore("todo");
var keyRange = IDBKeyRange.lowerBound(0);
var cursorRequest = store.openCursor(keyRange);
cursorRequest.onsuccess = function(e) {
var result = e.target.result;
if(!result == false)
return;
renderAll(result.value);
result.continue();
};
```

La función getAllTodoItems junta toda la información para la visualización de los datos. Después de la definición de algunas variables, se establece una transacción en modo lectura/escritura hacia el

Object Store todo. La variable keyrange contiene las claves de los datos creados anteriormente por el usuario (IDBKeyRange.lowerBound(0)), con un timestamp menor o igual al del dato añadido. Una consulta obtiene la información de los datos del Object Store listados (store.openCursor(keyRange)). En caso de éxito, se llama a la función renderTodo pasando como argumento el valor almacenado en el dato (result.value). Los otros valores de la lista keyRange se tratan por result.continue().

```
function renderTodo(row) {
var todos = document.getElementById("todoItems");
var li = document.createElement("li");
var a = document.createElement("a");
var t = document.createTextNode(row.text);
a.addEventListener("click", function() {
html5.indexedDB.deleteTodo(row.timeStamp);
}, false);
a.textContent = " [Eliminar]";
li.appendChild(t);
li.appendChild(a);
todos.appendChild(li)
}
```

La función renderTodo se ocupa de la visualización de los elementos codificados en el Object Store. Esto se hace usando el JavaScript clásico que nos parece inútil detallar. Observe simplemente la parte del script relativa al enlace (a) **Eliminar**.

```
a.addEventListener("click", function() {
html5.indexedDB.deleteTodo(row.timeStamp);
}, false);
```

Al hacer clic en el enlace, se llama a la función deleteTodo que aquí hemos visto anteriormente. El script pasa como argumento el timestamp del elemento llamado aquí row.

```
function addTodo() {
var all = document.getElementById("todo");
html5.indexedDB.addAll(todo.value);
todo.value = "";
}
```

Añade el valor de la zona de texto al Indexed Database.

Presentación y objetivo

Este API permite que cualquier contenedor de una página Web sea directamente modificable por el usuario sin tener que usar de forma obligatoria registros o zonas de texto. El internauta tiene la impresión de estar frente a un software de tratamiento de texto, más que frente a una página Web.

Hacer editable un contenido no es más que un agradable gadget si no podemos recuperar la codificación del usuario. Esto se puede hacer con PHP, llamando al servidor mediante un código del tipo:

```
<script type="text/javascript">
$.post('save.php', {
content: $('#content').html()
}
</script>
```

Pero también se conservan las malas costumbres. De hecho, con el API Storage (sessionStorage o localStorage), es posible utilizar JavaScript para almacenar estas modificaciones sin llamar al servidor. Se propone un ejemplo en la sección Guardar las modificaciones con el API Storage de este capítulo.

JavaScript siempre ha tenido la vocación de aumentar la interactividad de las páginas Web. El contenido editable, asociado al API Storage, vendrá a reforzar considerablemente esta interactividad.

Con los comandos asociados a los contenidos editables (cf. sección Los comandos del API de este capítulo), HTML5 permite o permitirá añadir un editor en línea sin tener que pasar por un framework, como por ejemplo jQuery o Dojo.



Este API se puede aplicar en local.

Disponibilidad

Esta edición de contenido se utiliza prácticamente en todos los navegadores. Mencionamos:

- Internet Explorer 5.5+.
- Firefox 3.6+.
- Google Chrome 4+.
- Safari 3.1+.
- Opera 9+.

Sin embargo, hay que destacar que no todas las propiedades y funcionalidades se incluyen forzosamente en los navegadores mencionados.

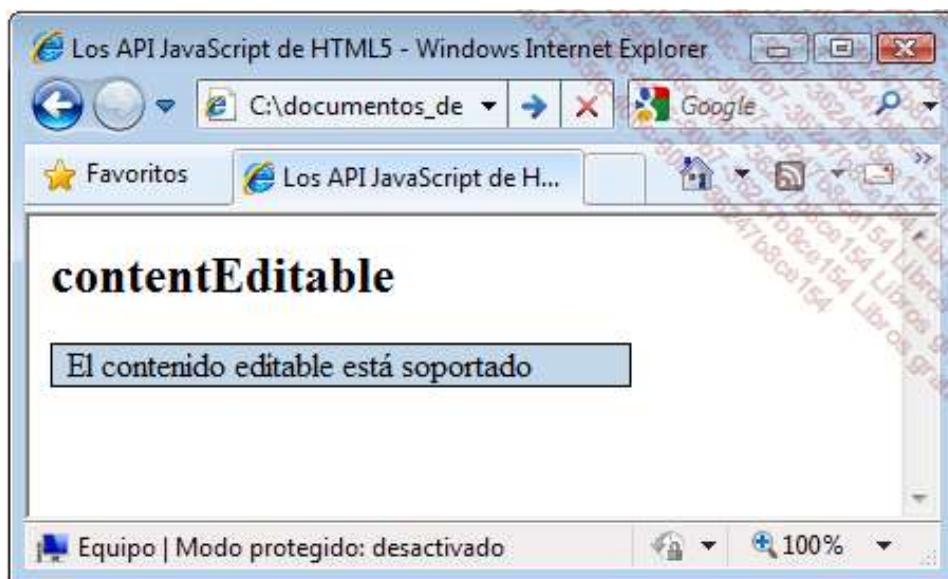
Los navegadores para Smartphone no son ajenos a este API. La edición de contenido está presente en:

- iOS Safari 5.0+.
- Android 3.0+.
- Opera Móvil 10.63+.

El mejor sistema para verificar que el contenido editable está soportado por el navegador es probar si la función se soporta por un elemento del documento. Esto se podría hacer con el siguiente código:

```
if(element.contentEditable != null)  
o  
if(typeof(element.contentEditable) != 'undefined')
```

Ejemplo



El código

```
<!DOCTYPE html>  
<html lang="es">  
<head>  
<title>Los API JavaScript de Html5</title>  
<meta charset=utf-8>  
<style type="text/css">
```

```
#box { width: 270px;
    border: 1px solid black;
    background-color: rgb(195,215,235);
    padding-left: 3px; }
</style>
<script type="text/javascript">
function init(){
if (document.getElementById("titulo").contentEditable != null) {
msg = "El contenido editable está soportado";
document.querySelector('#box').innerHTML = msg;
}
else {
alert("Su navegador no soporta la edición del contenido ");
}
}
</script>
</head>
<body onload="init();">
<h2 id="titulo">contentEditable</h2>
<div id="box">&nbsp;</div>
</body>
</html>
```

Comentario

```
if (document.getElementById("titulo").contentEditable != null) {
```

El script ha comprobado simplemente si la etiqueta de título <h2> identificada por el id `titulo` acepta esta funcionalidad.

Hacer que un elemento sea editable

1. El atributo contentEditable

Para hacer que un contenido sea editable, es suficiente con añadir el atributo `contentEditable` a una etiqueta de tipo bloque.

Los diferentes valores son:

`contentEditable="true"`

El elemento es editable. Si el atributo `contentEditable` se usa solo, toma el valor `true`.

`contentEditable="false"`

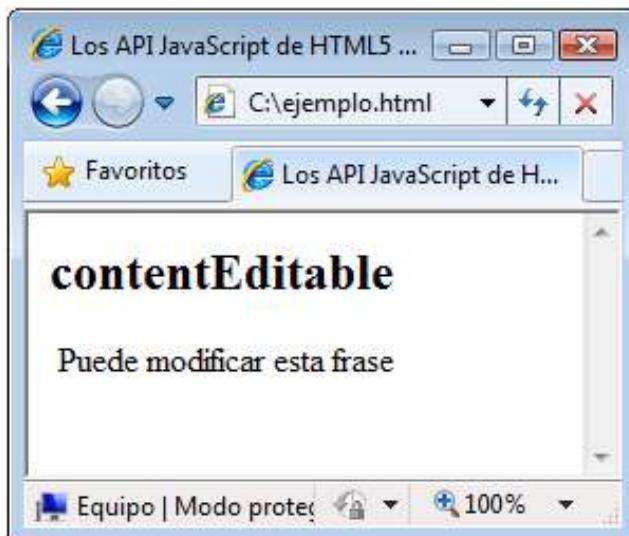
El elemento no es editable.

`contentEditable="inherit"`

Toma el valor del elemento padre.

Ejemplo

Hagamos que una capa sea editable.



Después de la edición:



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
</head>
<body>
<h2>contentEditable</h2>
<div contentEditable>Puede modificar esta frase
</div>
</body>
</html>
```

2. El atributo designMode

El atributo `designMode` hace todo el documento editable.

Los valores pueden ser:

`designMode="on"`

El documento es editable. Si el atributo `designMode` se usa solo, toma el valor `on`.

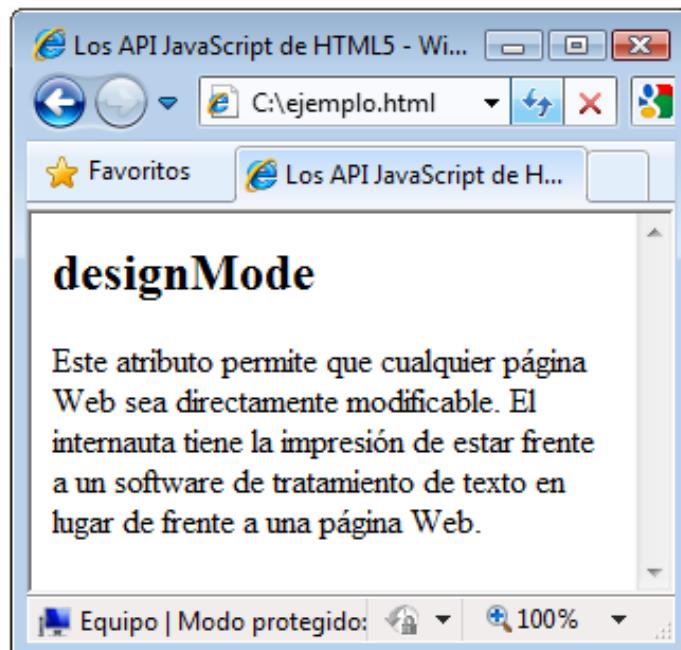
`designMode="off"`

El documento no es editable.

Ejemplo

Hagamos que toda la página sea editable. Veremos que el resultado final no tiene nada que ver con el documento inicial.

Antes:



Después:



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#box { width: 270px; }
</style>
<script type="text/javascript">
function load(){
window.document.designMode = "On";
}
</script>
</head>
<body contentEditable="true" onload="load()">
<h2>designMode</h2>
<div>
Este atributo permite que cualquier página Web sea directamente modificable. El internauta tiene la impresión de estar frente a un software de tratamiento de texto en lugar de frente a una página Web.
</div>
</body>
</html>
```

Comentario

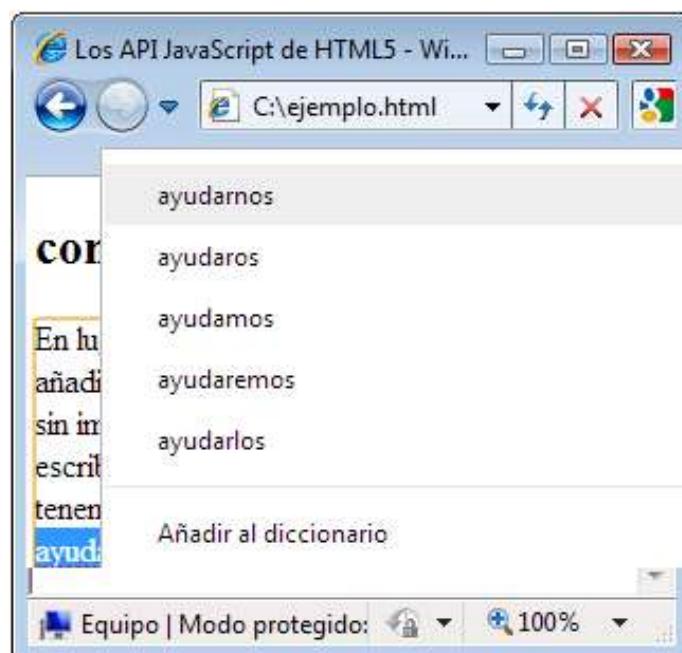
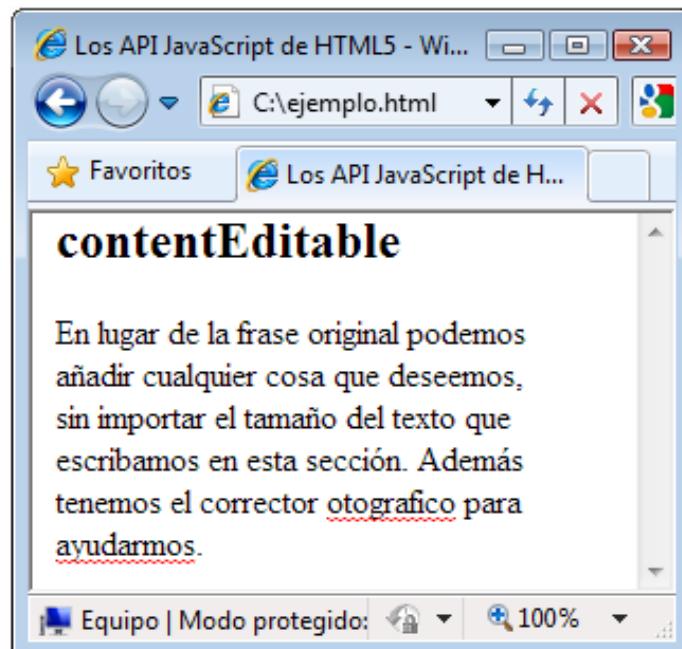
```
function load() {
window.document.designMode = "On";
}
```

Para hacer editable toda la página, hay que pasar por una función JavaScript.

3. El atributo spellcheck

El atributo `spellcheck` permite introducir un corrector ortográfico para el contenido codificado por el usuario en una zona editable.

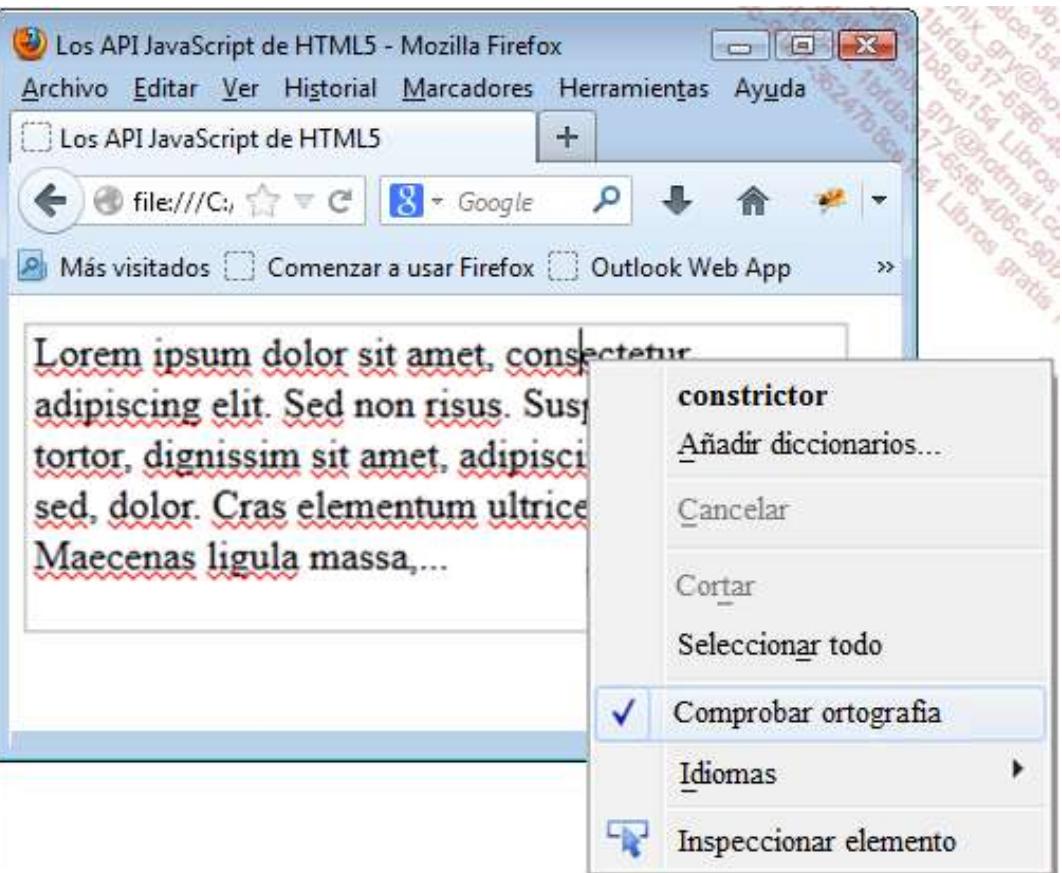
Ejemplo



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
</head>
<body>
<h2>contentEditable</h2>
<div id="box" contentEditable spellcheck>Puede modificar esta
frase.
</div>
</body>
</html>
```

Firefox permite activar o desactivar esta función de corrección de ortografía.

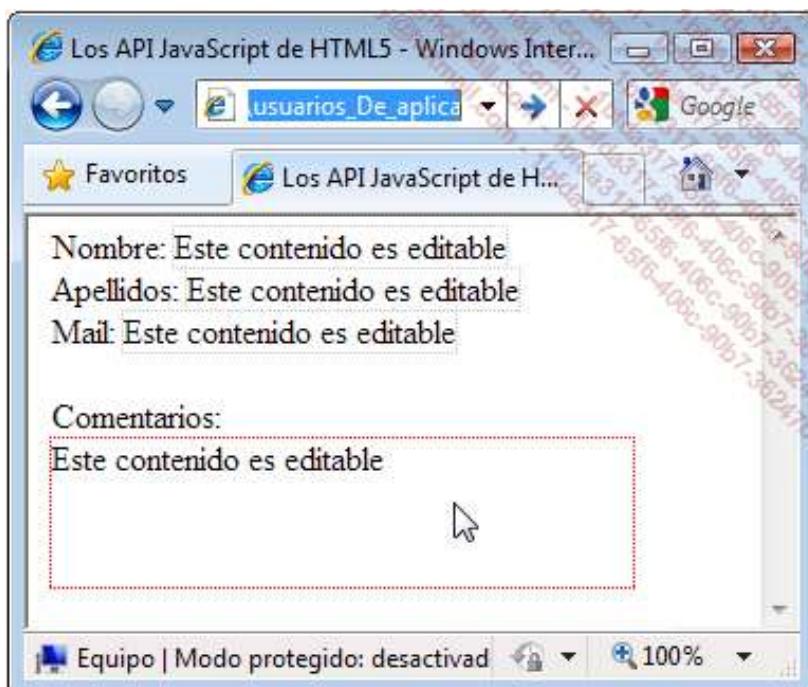


Señalar las zonas editables

Estas zonas editables son seguramente una novedad interesante, pero tienen el riesgo de desorientar al usuario. En lugar de ver la página salpicada de anotaciones como «Codifique lo que deseé» o «Puede modificar esto», algunos desarrolladores preferirían resaltarlos e imponer la convención de rodearlos de una discreta línea punteada.

Ejemplo

Rodeamos las zonas editables con una discreta línea punteada. Cuando se acceda a esta zona, la línea punteada se destaca en rojo.



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
[contenteditable] { outline: 1px dotted #cccccc; }
[contenteditable]:hover { outline: 1px dotted #ff0000; }
#box { width: 275px;
       height: 70px; }
</style>
</head>
<body>
Nombre: <span contenteditable>Este contenido es editable</span><br>
Apellidos: <span contenteditable>Este contenido es editable</span><br>
Mail: <span contenteditable>Este contenido es editable</span><br>
&nbsp;<br>
Comentarios:
<div id="box" contenteditable>
Este contenido es editable
</div>
</body>
</html>
```

Los comandos del API

Los comandos para interactuar con el documento son:

document.execCommand	Ejecuta un comando concreto.
document.queryCommandEnabled	Determina si un comando concreto se puede ejecutar en el documento en su estado actual.
document.queryCommandIndeterm	Determina si la selección actual está en un estado indeterminado.
document.queryCommandState	Determina si un comando dado se ha ejecutado en la selección actual.
document.queryCommandValue	Determina el valor actual del documento o del elemento seleccionado actual para un comando concreto.

No todos estos comandos están ya implementados en los navegadores, incluso en los más recientes.

Nos dedicamos al comando `document.execCommand`, que permite ejecutar un comando concreto.

La sintaxis es:

```
document.execCommand(comando, IHM, valor)
```

donde

- comando es el nombre del comando que se va a ejecutar.
- IHM indica si es necesaria una Interfaz Hombre-Máquina (opcional).
- valor: un eventual argumento de comando (opcional).

`document.execCommand` ofrece una lista impresionante de comandos:

backColor	Cambia el color de fondo del documento. El color se proporciona como argumento.
bold	Pone en negrita el elemento seleccionado.
contentReadOnly	Pone el contenido del documento en modo de solo lectura o editable, gracias al valor true o false del argumento.
copy	Copia el elemento seleccionado.
createLink	Crea un link para el elemento seleccionado.
cut	Corta el elemento seleccionado.
decreaseFontSize	Disminuye el tamaño de letra.
delete	Elimina el elemento seleccionado.
enableInlineTableEditing	Activa o desactiva la inserción de registros o columnas de una matriz.
enableObjectResizing	Activa o desactiva el redimensionamiento de las imágenes u otros elementos redimensionables.
fontName	Cambia el tipo de letra. El tipo de letra (por ejemplo, "Arial") se proporciona como argumento al comando.

	argumento al comando.
fontSize	Cambia el tamaño de letra. El tamaño (1-7) es un argumento.
foreColor	Cambia el color de letra. El color es un argumento.
formatBlock	Hace que el elemento seleccionado sea un elemento bloque.
heading	Añade un título. Es necesario dar el nombre de la etiqueta (por ejemplo, "h1" o "h6") como argumento.
hiliteColor	Cambia el color de fondo del elemento seleccionado. El valor del color es un argumento.
increaseFontSize	Aumenta el tamaño de letra.
indent	Mueve a la derecha el elemento seleccionado.
insertBrOnReturn	Pulsando la tecla [Intro], se inserta una etiqueta o un elemento bloque se divide en dos.
insertHorizontalRule	Inserta una línea horizontal.
insertHTML	Inserta código Html. Html (válido) es un argumento.
insertImage	Inserta una imagen. La dirección de la imagen (src) es un argumento.
insertOrderedList	Inserta una lista numerada.
insertUnorderedList	Inserta una lista de opciones.
insertParagraph	Inserta un párrafo alrededor del elemento seleccionado o de la línea actual.
italic	Pone el elemento seleccionado en itálica.
justifyCenter	Justifica al centro.
justifyleft	Justifica a la izquierda.
justifyright	Justifica a la derecha.
outdent	Ejecuta una sangría a la izquierda del elemento seleccionado.
paste	Copia un elemento cortado o copiado.
redo	Vuelve a hacer después de una anulación (ver undo).
removeFormat	Borra el formato del elemento seleccionado.
selectAll	Selecciona todo.
strikeThrough	Tacha la selección.
subscript	Utiliza la selección como subíndice.
superscript	Utiliza la selección como superíndice.
underline	Subraya la selección.
undo	Anula el último comando realizado.
unlink	Retira el link de un enlace
styleWithCSS	Sustituye el comando useCSS (sin uso). Con el

argumento true, modifica o genera los atributos de estilo en la etiqueta; con el argumento false genera los elementos de formato.

Aquí están todos los elementos para hacer estas zonas editables en un editor Html.

De momento, la compatibilidad de estos comandos está lejos de ser perfecta, pero este API todavía se halla en plena evolución. Se recomienda consultar el artículo <http://www.quirksmode.org/dom/execCommand.html>

Ejemplo

Elaboramos un pequeño editor en línea que utiliza algunos de los comandos detallados anteriormente.

Si lo desea, puede echar un vistazo al ejemplo completo en la dirección: <http://www.quirksmode.org/dom/execCommand/>

Situación inicial:



Después de usarlo:



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
.boutons { margin-bottom: 5px; }
#box { width: 320px;
       height: 120px;
       border: 1px solid silver;
       margin-top: 20px;
       padding-left: 3px; }
form { margin-top: 15px; }
input { -webkit-border-radius: 18px;
        -moz-border-radius: 18px;
        border-radius: 18px;
        background-color: #b6d3f4;
        background-image: -webkit-gradient(linear, left top, left
bottom, from(#b6d3f4), to(#5483b8)) !important;
        background-image: -webkit-linear-gradient(top, #b6d3f4,
#5483b8) !important;
        background-image: -moz-linear-gradient(top, #b6d3f4,
#5483b8) !important;
        background-image: -ms-linear-gradient(top, #b6d3f4,
#5483b8) !important;
        background-image: -o-linear-gradient(top, #b6d3f4,
#5483b8) !important;
        box-shadow: inset 0 1px 1px white, 0 3px 3px
rgba(0,0,0,0.6);
        text-shadow: 0 -1px 0 rgba(0,0,0,0.3);
        border: 1px solid #9ac9ff !important;
        font: 12px, Sans-Serif;
        font-weight: bold;
        color: white !important;
        position: relative; }
```

```

</style>
</head>
<body>
<form>
<div class="botones">
<input type="button" value="N"
onclick="document.execCommand('bold' , false, '')">
<input type="button" value="I"
onclick="document.execCommand('italic', false, '')"
style="font-style:italic">
<input type="button" value="S"
onclick="document.execCommand( 'underline', false, '')"
style="text-decoration:underline">
<input type="button" value="A"
onclick="document.execCommand('strikeThrough', false, '')"
style= "text-decoration:line-through">
<input type="button" value="++"
onclick="document.execCommand('increaseFontSize', false, '')">
<input type="button" value="--"
onclick="document.execCommand('decreaseFontSize', false, '')">
<input type="button" value="Borrar"
onclick="document.execCommand('delete', false, '')">
</div>
<div class="botones">
<input type="button" value="Izquierda"
onclick="document.execCommand('justifyLeft', false, '')">
<input type="button" value="Centrado"
onclick="document.execCommand('justifyCenter', false, '')">
<input type="button" value="Derecha"
onclick="document.execCommand('justifyRight', false, '')">
<input type="button" value="Cancelar"
onclick="document.execCommand('undo', false, '')">
</div>
<div class="boutons">
<input type="button" value="Exp"
onclick="document.execCommand('superscript', false, '')">
<input type="button" value="Ind"
onclick=
"document.execCommand('subscript', false, '')">
<input type="button" value="Cortar"
onclick="document.execCommand('cut', false, '')">
<input type="button" value="Copiar"
onclick="document.execCommand('copy', false, '')">
<input type="button" value="Pegar"
onclick="document.execCommand('paste', false, '')">
</div>
</form>
<div id="box" contentEditable>
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed non
risus. Suspendisse lectus tortor, dignissim sit amet, adipiscing
nec, ultricies sed, dolor. Cras elementum ultrices diam. Maecenas
ligula massa, ...
</div>
</body>
</html>

```

Comentario

```

<input type="button" value="G" onclick="document.execCommand('bold' ,
false, '')">

```

Al hacer clic en el documento (onclick()), document.execCommand, se usa el comando bold para poner en negrita el texto seleccionado.

Los otros botones se construyen siguiendo el mismo esquema.

Guardar las modificaciones con el API Storage

Recordamos que dejar la posibilidad al usuario de modificar el contenido es de una utilidad limitada si no guardamos las modificaciones que hace. El atributo contentEditable no permite al usuario efectuar por sí mismo esta operación. El almacenamiento de estos datos nuevos se puede hacer sin llamar al servidor, usando el API JavaScript Storage de HTML5.

Ejemplo

Pasamos los datos codificados por el usuario de una página a otra página.

Consideramos la página inicial (pagina1.htm):



El usuario rellena los datos y se almacenan en el espacio localStorage.



En otra página del sitio Web (pagina2.htm), es suficiente con recuperar los valores guardados en localStorage.



El código (pagina1.htm)

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
```

```
<meta charset=utf-8>
<style type="text/css">
[contenteditable] { outline: 1px dotted #969696;}
[contenteditable]:hover { outline: 1px dotted #ff0000;}
a { color: black;}
form.css fieldset { padding: 1em;
                     width: 300px;
                     border:1px solid black;}
form.css label { display: block; float: left;
                  width: 90px;}
.long { width: 200px;
        display: block;
        float: left;
        padding-left: 5px;
        color: rgb(150,150,150);
        margin-bottom: 5px;}
h1 { margin-bottom: 12px;}
</style>
<script type="text/javascript">
function set_item() {
var nombre = "nombre";
var dato1 = document.getElementById("f1").innerHTML
localStorage.setItem(nombre, dato1);
var apellidos = "apellidos";
var dato2 = document.getElementById("f2").innerHTML
localStorage.setItem(apellidos, dato2);
var dirección = "dirección";
var dato3 = document.getElementById("f3").innerHTML
localStorage.setItem(dirección, dato3);
var cp = "cp";
var dato4 = document.getElementById("f4").innerHTML
localStorage.setItem(cp, dato4);
var ciudad = "ciudad";
var dato5 = document.getElementById("f5").innerHTML
localStorage.setItem(ciudad, dato5);
var país = "país";
var dato6 = document.getElementById("f6").innerHTML
localStorage.setItem(país, dato6);
}
</script>
</head>
<body>
<h1>Página 1</h1>
<form class="css">
<fieldset>
<div><label>Nombre: </label>
<div id="f1" contentEditable class="long">Zona editable</div>
</div>
<div><label>Apellidos: </label>
<div id="f2" contentEditable class="long">Zona editable</div>
</div>
<div><label>Dirección: </label>
<div id="f3" contentEditable class="long">Zona editable</div>
</div>
<div><label>Código Postal: </label>
<div id="f4" contentEditable class="long">Zona editable</div>
</div>
<div><label>Ciudad: </label>
<div id="f5" contentEditable class="long">Zona editable</div>
</div>
<div><label>País: </label>
<div id="f6" contentEditable class="long">Zona editable</div>
</div>
</fieldset>
</form>
<p><a href="pagina2.htm" onclick="set_item()">Siga</a>
```

```
su pedido.</p>
</body>
</html>
```

Comentario

Mencionemos únicamente:

```
var nombre = "nombre";
var dato1 = document.getElementById("f1").innerHTML
localStorage.setItem(nombre, dato1);
```

El dato está contenido en la variable var nombre = "nombre". El valor es el de la capa editable (document.getElementById("f1").innerHTML). Se almacena la pareja clave/valor (localStorage.setItem(nombre, dato1)). Y así sucesivamente para el resto de las entradas.

El código (pagina2.htm)

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
div { border: 1px solid black;
      font-size:16px;
      font-weight: bold;
      width: 250px;
      padding-left: 3px; }
</style>
<script type="text/javascript">
function get_item() {
document.getElementById("f1").innerHTML =
localStorage.getItem("nombre");
document.getElementById("f2").innerHTML =
localStorage.getItem("apellidos");
document.getElementById("f3").innerHTML =
localStorage.getItem("direccion");
document.getElementById("f4").innerHTML =
localStorage.getItem("cp");
document.getElementById("f5").innerHTML =
localStorage.getItem("ciudad");
document.getElementById("f6").innerHTML =
localStorage.getItem("pais");
}
</script>
</head>
<body onload="get_item()">
<h1>Página 2</h1>
<form>
_N?uj
<span id="f6"></span>
</div>
</form>
</body>
</html>
```

Comentario

El valor almacenado se recupera con localStorage.getItem("nombre"), y así sucesivamente para los otros datos.

Presentación y objetivos

La consulta de las páginas en modo desconectado está soportada por el API Offline Application Caching, pero lo veremos en la Red expresiones como Application Cache o AppCache.

HTML5 ha creado un dispositivo para poder consultar las aplicaciones Web en modo desconectado. Con este API, los archivos se almacenan en la caché del navegador y la aplicación continúa siendo consultable incluso cuando el usuario está desconectado. Si una página o una aplicación Web está en modo desconectado, gracias a este API el navegador puede utilizar (de manera transparente) la copia que hay en caché.

Todos los navegadores ofrecen de alguna forma el cacheo de las aplicaciones, pero el funcionamiento del API AppCache es diferente del proceso normal de cacheo que hace el navegador. La caché del navegador contiene por lo común las páginas y sus recursos asociados que previamente hemos visitado. El API AppCache permite al desarrollador especificar que las páginas y sus recursos (archivos js o css, imágenes, etc.) se tienen que cachear.

Las aplicaciones Web que continúan funcionando incluso sin conexión a Internet no tienen mucho sentido con la calidad actual de las conexiones fijas que existen en los domicilios particulares. Por el contrario, si consulta la Red desde su Smartphone, mientras pasa por un túnel, viaja en metro, en un ascensor o se encuentra en una zona sin cobertura 3G, corre el riesgo de que la conexión se corte de manera frecuente y, con ella, la aplicación que está consultando. Por tanto, este API *offline* está destinado principalmente a los teléfonos portátiles, que son susceptibles de sufrir más regularmente problemas de desconexión. Estamos, de alguna manera, frente a aplicaciones «híbridas» que funcionan al mismo tiempo en línea (*online*) y en modo desconectado (*offline*).

Esta navegación *offline* tiene las siguientes ventajas:

- Siempre está disponible, haya o no una conexión a Internet.
- Es rápida porque los archivos están cacheados localmente y se cargan con mayor rapidez.
- Reduce la carga del servidor al reducir las consultas que se hacen a este.

Seguridad

Solo Firefox pide su autorización para cachear los datos en caso de uso en modo desconectado.



Los otros navegadores no avisan del cacheo de los archivos y lo hacen de manera automática y transparente para el usuario.

Depuración

Cuando depuramos y probamos una aplicación Web que utiliza el cacheado, es bueno partir de una situación correcta. Para conseguirlo, se aconseja vaciar la caché al inicio o durante una actualización.

Firefox: **Herramientas - Opciones - Avanzado - pestaña Red**

Opciones

General Pestañas Contenido Aplicaciones Privacidad Seguridad Sync Avanzado

General Elección de datos Red Actualizar Cifrado

Conexión

Configurar cómo Firefox se conecta a Internet [Configuración...](#)

Contenido web cacheado

El contenido web está actualmente usando 16,5 MB de espacio en disco [Limpiar ahora](#)

Ignorar la administración automática de caché

Limitar la caché a MB de espacio

Contenido web sin conexión y datos de usuario

Las aplicaciones están actualmente usando 0 bytes de espacio en disco de caché [Limpiar ahora](#)

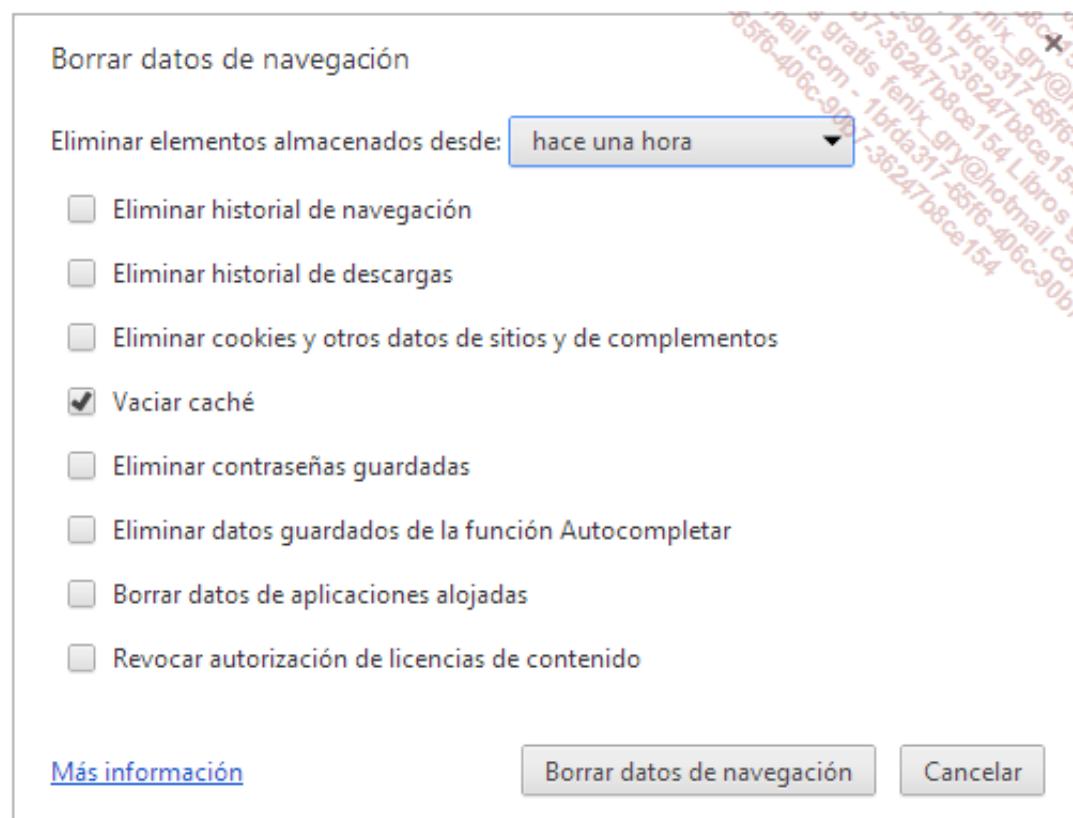
Avisarme si un sitio solicita guardar datos para uso sin conexión [Excepciones...](#)

Los siguientes sitios web tienen permiso para guardar datos para el uso en modo sin conexión:

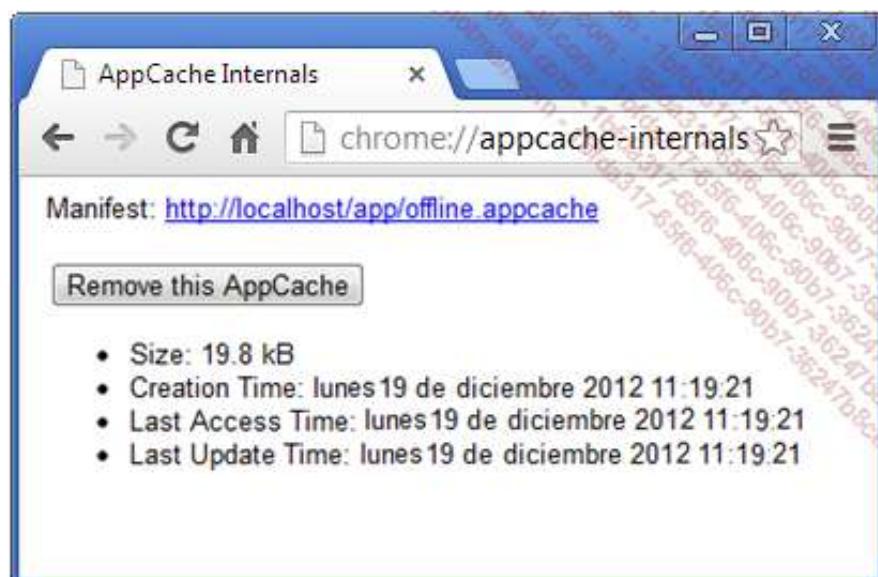
[Eliminar...](#)

Aceptar Cancelar Ayuda

Chrome: Herramientas  - Historial - Borrar datos de navegación



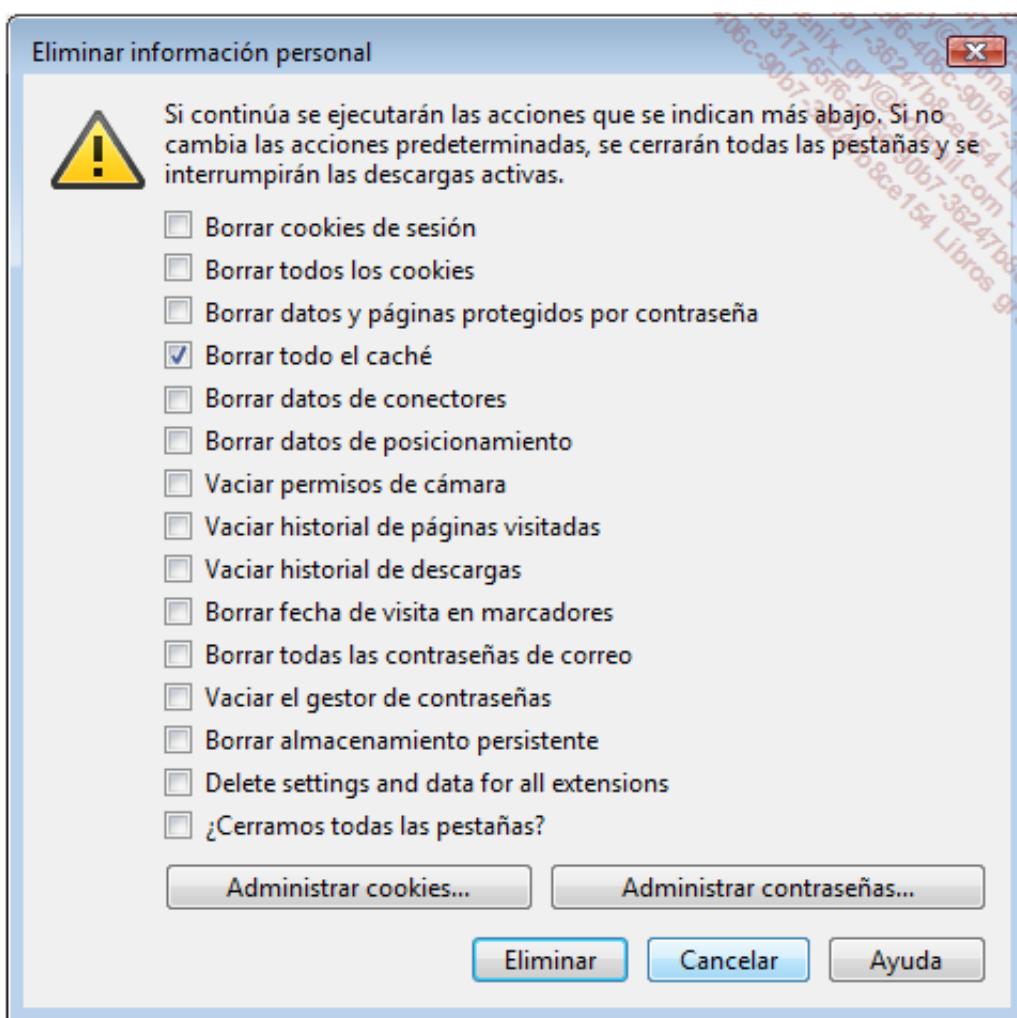
Chrome puede eliminar solo las aplicaciones cacheadas específicas, codificando `chrome://appcache-internals` en la barra de dirección.



Safari: Herramientas  - Restaurar Safari



Opera: Configuración - Eliminar información personal - Borrar todo el caché



Los ejemplos y pruebas solo funcionarán a través de un servidor local o en línea.

Disponibilidad del API

Este API es reconocido por los siguientes navegadores:

- Internet Explorer 10+
- Firefox 3.6+
- Safari 4+
- Google Chrome 5+
- Opera 10.6+

Respecto a los Smartphone o tabletas, el API está disponible para los siguientes navegadores:

- iOS Safari 3.2+
- Android 2.1+
- Opera Móvil 11+

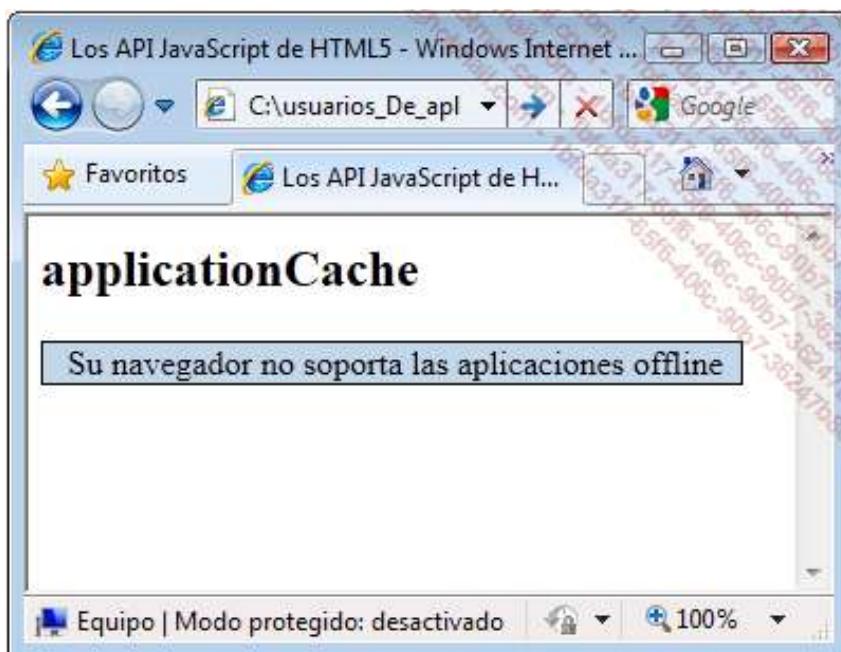
Para probar la disponibilidad del API, verificamos la presencia del objeto `applicationCache` de `window`.

Se puede utilizar el siguiente código:

```
if(window.applicationCache) {  
// Su navegador soporta las aplicaciones offline  
}  
  
o  
  
if (!window.applicationCache){  
alert("Su navegador no soporta las aplicaciones  
offline");  
}
```

Ejemplo

Probamos la disponibilidad en Internet Explorer 8.



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#box { border: 1px solid black;
        background-color: rgb(195,215,235);
        padding-left: 3px;
        text-align: center; }
</style>
<script type="text/javascript">
function init(){
if (!window.applicationCache){
msg = "Su navegador no soporta las aplicaciones offline";
document.querySelector('#box').innerHTML = msg;
}
else {
msg = "Su navegador soporta las aplicaciones offline";
document.querySelector('#box').innerHTML = msg;
}
}
</script>
</head>
<body onload="init();">
<h2>applicationCache</h2>
<div id="box">&nbsp;</div>
</body>
</html>
```

Para este API, también puede ser útil probar la disponibilidad de la conexión.

Conectado:

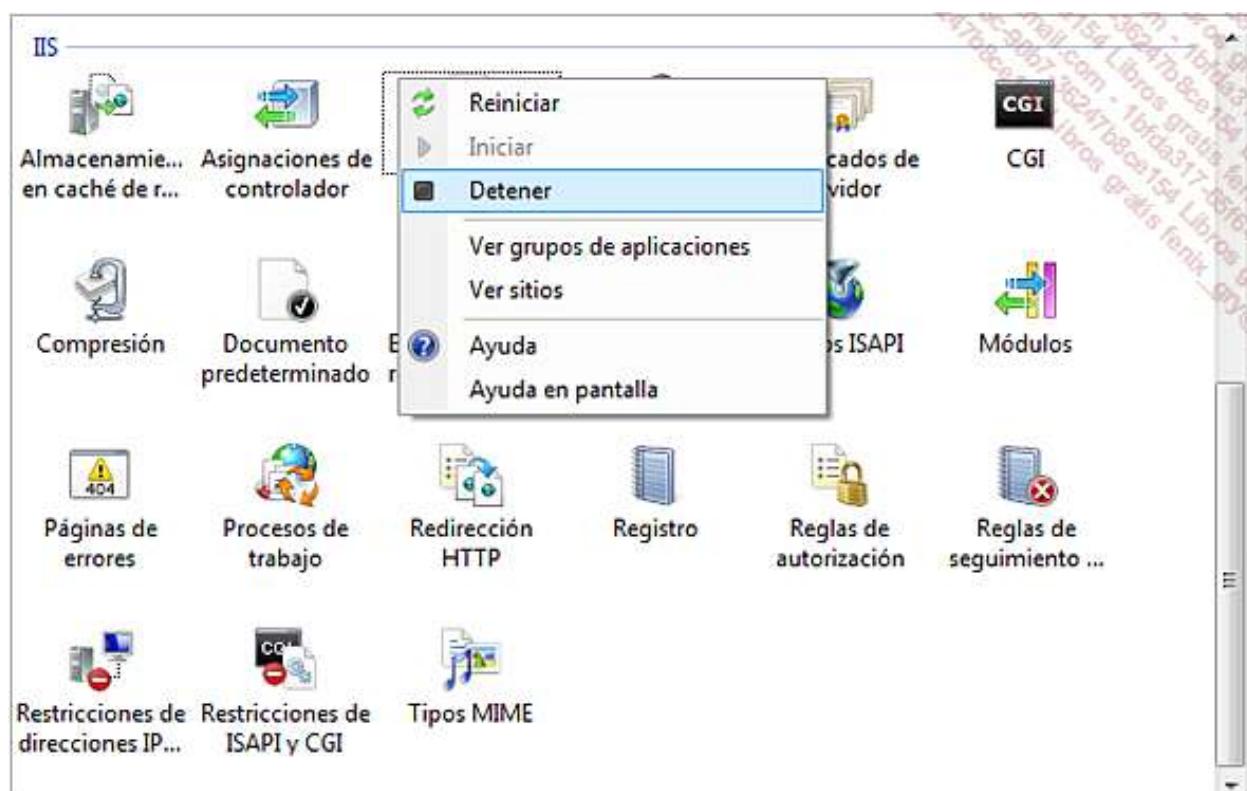


Desconectado:



Para probar en modo desconectado una aplicación que reside en un servidor remoto, es suficiente con interrumpir su conexión a Internet y recargar o actualizar la página.

Si prueba una aplicación que reside en un servidor local, hay que detener el funcionamiento del servidor local y recargar la página.



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#box { width: 80px;
       border: 1px solid black;
       text-align: center;
       margin: auto;}
```

```

.offline { background: #c00; }
.online { background: #0c0; }
</style>
<script type="text/javascript">
function init(){
if (!window.applicationCache) {
msg = "El API Offline no está soportado";
document.querySelector('#box').innerHTML = msg;
}
if (window.navigator.onLine) {
document.querySelector('#box').className= 'online';
msg = "Conectado";
document.querySelector('#box').innerHTML = msg;
} else {
document.querySelector('#box').className= 'offline';
msg = "Desconectado";
document.querySelector('#box').innerHTML = msg;
}
}
</script>
</head>
<body onload="init();">
<h2>applicationCache</h2>
<div id="box">&nbsp;</div>
</body>
</html>

```

Comentario

También se pueden utilizar manejadores de eventos DOM 2.

```

window.addEventListener("offline", function(e) {
alert("offline");
}, false);
window.addEventListener("online", function(e) {
alert("online");
}, false);

```

El archivo de cacheado (manifest)

La gestión del cacheado de archivos está soportada por un archivo llamado manifiesto (*manifest*), que es un sencillo archivo de texto plano en UTF-8. Permite al desarrollador especificar los archivos y recursos que el navegador debe cachear. Una vez en la caché, el usuario localizará estos archivos en local sin depender del servidor o de la conexión.

El navegador descarga los recursos solo cuando se modifican o actualizan.

Observe que este archivo de manifiesto:

- Tiene la extensión .appcache (text/cache-manifest). W3C recomienda esta extensión desde el 15 de junio del 2011. Atención, existe la posibilidad de encontrar en la Red tutoriales que usan la antigua nomenclatura, .manifest.
- Tiene que garantizar que su servidor o el servidor que alberga su aplicación reconoce el tipo Mime-Type.
- El archivo de manifiesto siempre empieza con CACHE MANIFEST (obligatoriamente en mayúsculas).
- Si son necesarios los comentarios, estos tienen que precederse por el signo #.
- Para que sea funcional, este manifiesto se debe declarar en el archivo Html: <html manifest= "nombre_del_archivo.appcache">
- Es posible añadir líneas vacías para la legibilidad del archivo.

El manifiesto tiene tres secciones: CACHE, NETWORK y FALLBACK.

CACHE	Lista los archivos y los recursos que hay que cachear.
NETWORK	Lista los recursos que solo están disponibles en línea (por ejemplo, un archivo PHP). Estos archivos no se cachearán.
FALLBACK	Archivos que se van a vizualizar en caso de que no estén disponibles los recursos solicitados. La sintaxis es:url_original url_archivo_fallbackUn joker con la notación / url_archivo_fallback devuelve todos los archivos no disponibles para el archivo que se indica.

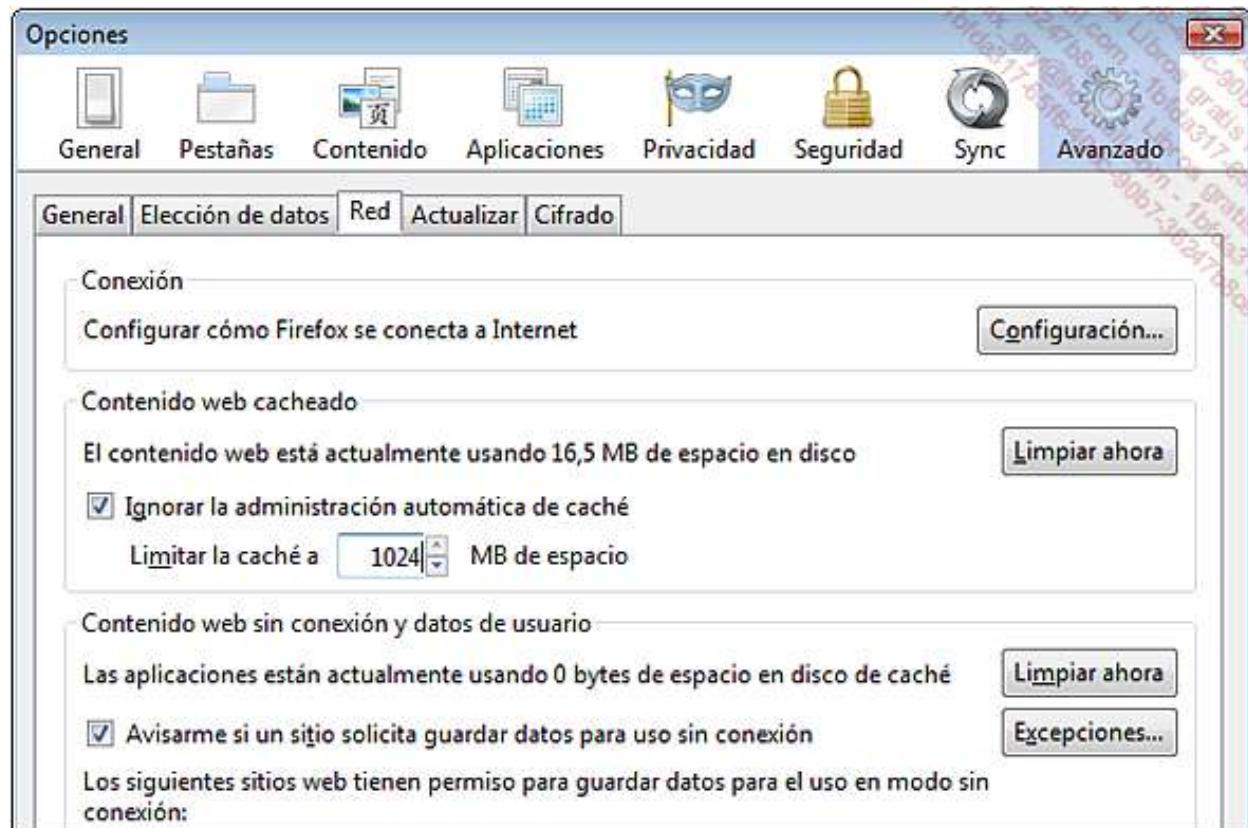
Observe que la URL que se usa en el manifiesto puede ser relativa o absoluta y que nada impide cachear los recursos situados en otro dominio diferente al de la página que declara el manifiesto.

¿Hay límites sobre la cantidad de información que se puede cachear? Varios navegadores imponen restricciones de tamaño diferentes en las aplicaciones fuera de línea.

Para empezar, los navegadores móviles tienden a ser restrictivos con el cacheado por motivos de espacio limitado en este tipo de periféricos. Actualmente, la versión de Safari que funciona con iPad y iPhone limita cada aplicación *offline* a 5 MB.

Los navegadores de escritorio son asombrosamente contradictorios.

En Firefox, una aplicación cacheada puede ocupar hasta 50 MB de espacio con los argumentos por defecto y el usuario puede aumentar este límite. Para aplicar los argumentos de cachado personalizados, en el menú de Firefox, seleccione **Opciones** - pestaña **Avanzado** - pestaña **Red**.



Para Google Chrome, el límite es (de momento) 5 MB.

Esta falta de coherencia entre los navegadores plantea un problema. Si desarrolla a partir de ahora una aplicación de más de 5 MB de datos, funcionará correctamente en Firefox, pero no en Chrome. No olvidemos que la implementación de los API JavaScript de Html5 está en plena evolución, incluso en fase experimental. Sin duda, estas aplicaciones fuera de línea dispondrán probablemente en el futuro de cantidades más grandes de espacio.

Un archivo de manifiesto puede tener la siguiente forma:

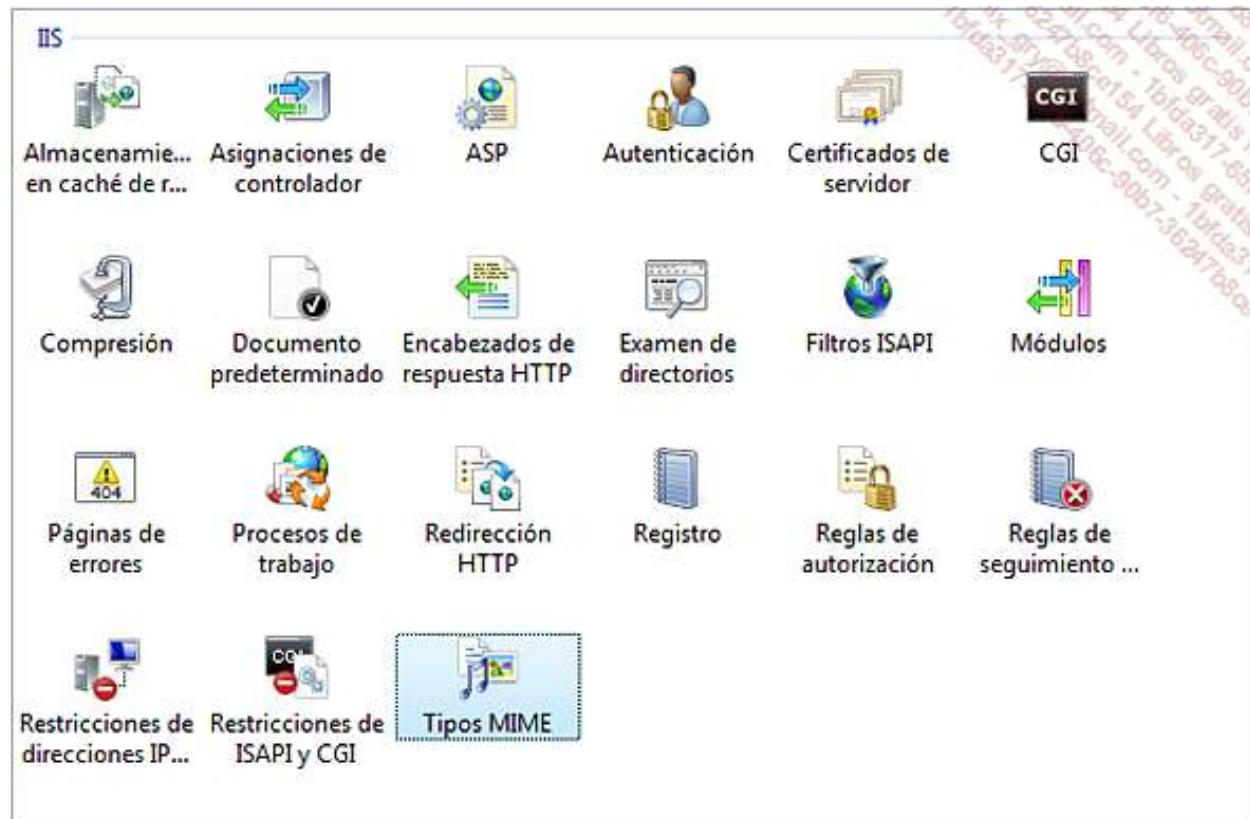
```
CACHE MANIFEST
# version 1.0
CACHE:
index.html
fallback.html
style.css
imagenes/1.png
imagenes/2.png
imagenes/3.png
imagenes/4.png
FALLBACK:
/ fallback.html
NETWORK:
network.html
```

Configurar IIS

Hay que configurar el servidor (local) para que reconozca este nuevo Tipo MIME.

Abra IIS Manager (`inetmgr` en la línea de texto de búsqueda de Windows 7).

En la pantalla que se presenta, elija **Tipos MIME**.

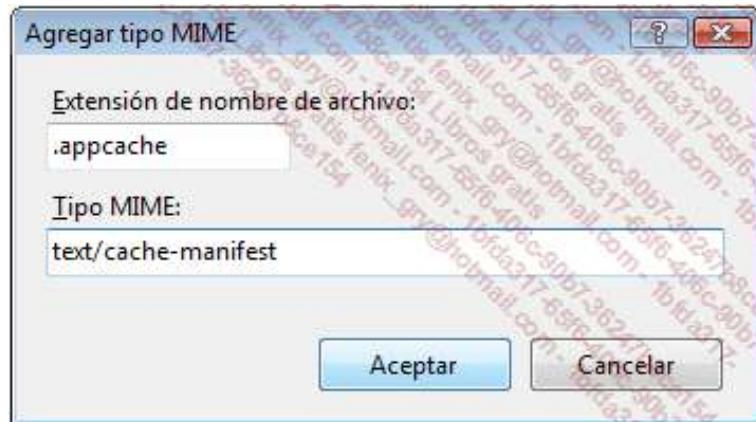


En la nueva pantalla, haga clic con el botón derecho del ratón y seleccione **Agregar**.

The screenshot shows the 'Tipos MIME' (MIME Types) management page. The title bar says 'Tipos MIME'. A tooltip for the 'Agregar...' button says 'Añadir un tipo MIME de contenido'. The main area has a table with columns 'Extensión', 'Tipo MIME', and 'Tipo de entrada'. The table lists several file extensions and their corresponding MIME types and entry types. A context menu is open over the last row of the table, showing options 'Agregar...', 'Ayuda', and 'Ayuda en pantalla'.

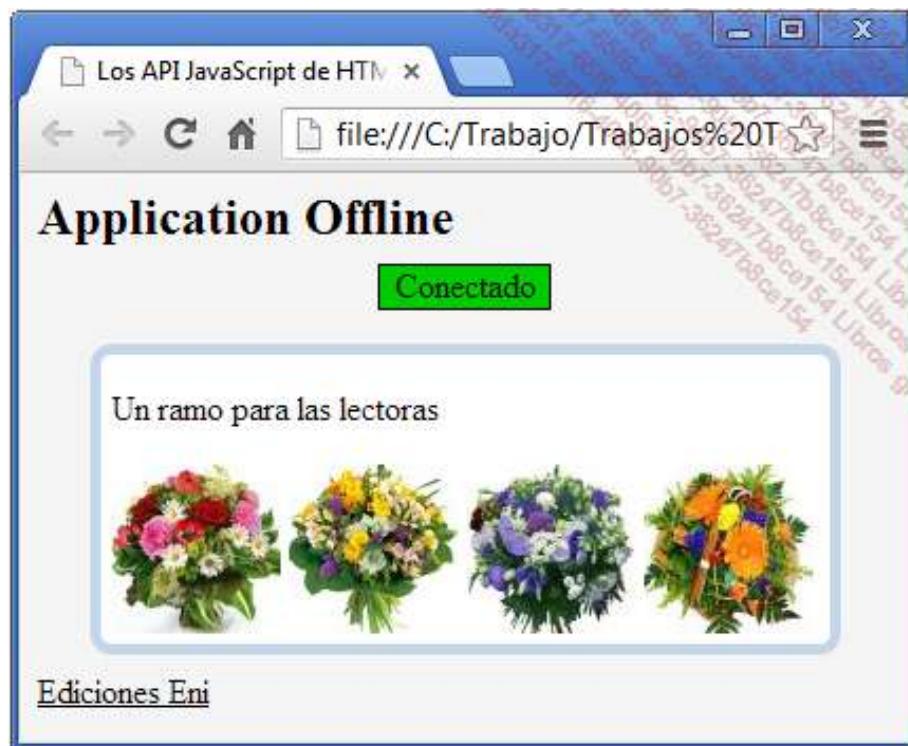
Extensión	Tipo MIME	Tipo de entrada
.323	text/h323	Local
.aaf	application/octet-stream	Local
.aca	application/octet-stream	Local
.accdb	application/msaccdb	Local
.accde	application/msaccde	Local
.accdt	application/msaccdt	Local
.acx	application/internetcdf	Local
.afm	application/octet-stream	Local

Codifique como extensión del archivo .appcache y como tipo MIME text/cache-manifest.



Ejemplo

Apliquemos todo esto con el siguiente ejemplo:



Las imágenes, como los archivos de los ejemplos, están disponibles en la página Información.

El archivo JavaScript externo (script.js)

```
function init(){
if (!window.applicationCache) {
msg = "El API offline no está soportado";
document.querySelector('#box').innerHTML = msg;
}
if (window.navigator.onLine) {
document.querySelector('#casilla').className= 'online';
msg = "Conectado";
document.querySelector('#casilla').innerHTML = msg;
} else {
document.querySelector('#casilla').className= 'offline';
msg = "Desconectado";
document.querySelector('#casilla').innerHTML = msg;
}
}
```

La hoja de estilo CSS externa (style.css)

```
a { color: black; }
body {background: #f5f5f5;}
#box{ margin: 0 auto;
      background: white;
      width: 340px;
      border:5px solid rgb(195,215,235);
      padding-left:5px;
      -webkit-border-radius: 10px;
      -moz-border-radius: 10px;
      border-radius: 10px;}
#casilla { width: 80px;
            border: 1px solid black;
            text-align: center;
            margin: auto;}
.parrafo { margin-top: 8px;
           margin-bottom: 8px;}
.offline { background: #c00;}
.online { background: #0c0;}
```

El archivo Html propiamente dicho (index.html)

```
<!DOCTYPE html>
<html lang="en" manifest="offline.appcache">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<link rel="stylesheet" href="style.css">
<script src="script.js"></script>
</head>
<body onload="init()">
<h2>Aplicación Offline</h2>
<div id="casilla">&nbsp;</div>
<p>
<div id="box">
<p class="parrafo">Un ramo para las lectoras</p>




</div>
</p>
<footer>
<p class="parrafo">
<a href=
" http://www.ediciones-eni.com/ /">Ediciones Eni</a></p>
</footer>
</body>
</html>
```

El archivo de fallback (fallback.html)

```
<!DOCTYPE html>
<html lang="es" manifest="offline.appcache">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style>
div { margin-top: 40px;
      margin-left: 40px;
      padding: 4px;
      width: 220px;
```

```

        border: 1px solid black;
        background: #1e8449;
    }

```

</style>

</head>

<body>

<h2>Application Offline</h2>

<div>

Lo sentimos, esta página no está prevista para su uso offline.

</div>

</body>

</html>

El archivo de manifiesto (offline.appcache)

Queremos que los archivos index.html, fallback.html, style.css, script.js y las imágenes ramo1.jpg, ramo2.jpg, ramo3.jpg y ramo4.jpg estén disponibles en modo desconectado.

El archivo de manifiesto offline.appcache se convierte en:

```

CACHE MANIFEST
# version 1.0
CACHE:
index.html
fallback.html
style.css
script.js
ramo1.jpg
ramo2.jpg
ramo3.jpg
ramo4.jpg
FALLBACK:
/ fallback.html
NETWORK:
*

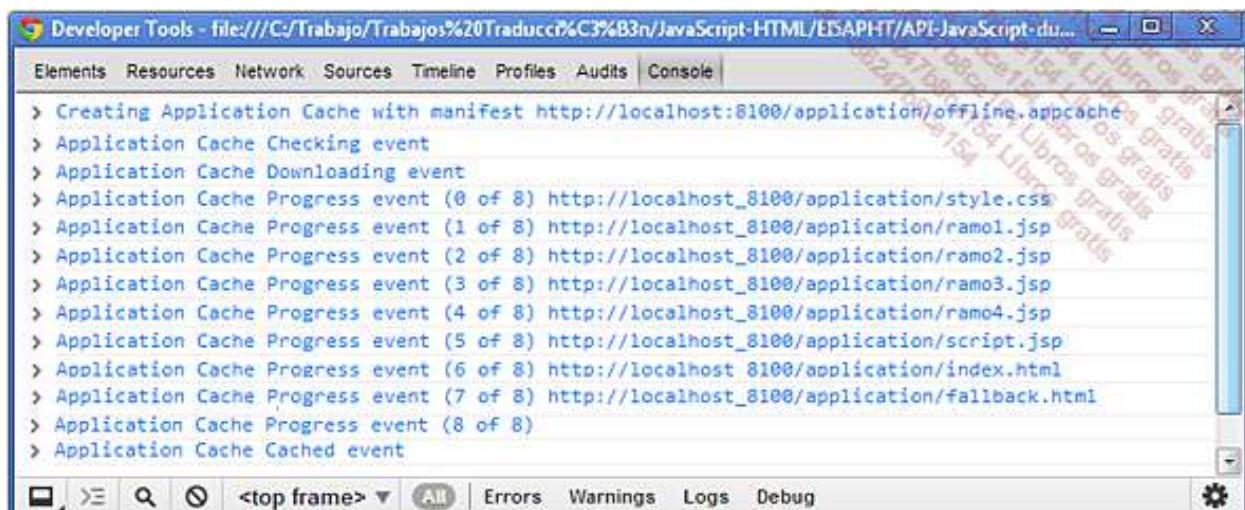
```

El asterisco que acompaña a la palabra clave NETWORK significa que todos los recursos que no se cachearán necesitan una conexión.

Al cargar la página, estos archivos se cachearán para su uso en modo desconectado.

Un vistazo a las herramientas de desarrollo de Chrome permiten comprobarlo.

Activando la pestaña **Console**:



Y con la pestaña **Resources**:

Elements	Resources	Network	Sources	Timeline	Profiles	Audits	Console
▼ Folders	Recurso						
▶ Folders (4_3_1.htm)	http://localhost:8100/application/fallback.html						
▼ Web SQL	http://localhost:8100/application/index.html						
▶ IndexedDB	http://localhost:8100/application/offline.appcache						
▶ Local Storage	http://localhost:8100/application/ramo1.jpg						
▶ Session Storage	http://localhost:8100/application/ramo2.jpg						
▶ Cookies	http://localhost:8100/application/ramo3.jpg						
▼ Application Cache	http://localhost:8100/application/ramo4.jpg						
localhost	http://localhost:8100/application/script.jsp						
	http://localhost:8100/application/style.css						

Firefox también nos proporciona información codificando `about:cache` en la barra de direcciones.

Offline cache device				
Key	Data size	Fetch count	Last modified	Expires
http://localhost:8100/application/offline.appcache	352 bytes	1	2013-04-14 20:46:47	2013-04-14 20:47:22
http://localhost:8100/application/index.html	704 bytes	1	2013-04-14 20:46:47	2013-04-14 20:47:14
http://localhost:8100/application/fallback.html	456 bytes	1	2013-04-14 20:46:47	2013-04-14 20:50:18
http://localhost:8100/application/style.css	507 bytes	1	2013-04-14 20:46:47	2013-04-14 20:47:29
http://localhost:8100/application/script.jsp	463 bytes	1	2013-04-14 20:46:47	2013-04-14 20:54:54
http://localhost:8100/application/ramo1.jsp	3664 bytes	1	2013-04-14 20:46:47	2013-04-14 20:52:45
http://localhost:8100/application/ramo2.jsp	3543 bytes	1	2013-04-14 20:46:47	2013-04-14 20:51:19
http://localhost:8100/application/ramo3.jsp	3763 bytes	1	2013-04-14 20:46:47	2013-04-14 20:59:23
http://localhost:8100/application/ramo4.jsp	3856 bytes	1	2013-04-14 20:46:47	2013-04-14 20:59:22

No daremos por acabado este ejemplo sin ilustrar el papel del archivo fallback durante una consulta fuera de línea. Si hacemos clic en el enlace **Ediciones Eni**, que no está disponible offline, se muestra el archivo fallback.

Los API JavaScript de HTM

file:///C:/Trabajo/Trabajos%20T

Application

Un ramo para



Ediciones Eni

www.ediciones-eni.com

Los API JavaScript de HTM

file:///C:/Trabajo/Trabajos%20T

Aplicación Offline

Lo sentimos, esta página no está prevista para su consulta offline.

Gestionar la caché

1. Los estados

Cada caché tiene un estado que indica su situación actual. Las cachés que comparten el mismo URL de manifiesto comparten el mismo estado, que es uno de los siguientes:

0	UNCACHED	Un valor especial que indica que un objeto ApplicationCache no se ha inicializado totalmente o ha fallado el cacheo.
1	IDLE	La caché no se está usando.
2	CHECKING	El manifiesto se está controlando para posibles actualizaciones.
3	DOWNLOADING	Los recursos se están descargando para añadirse a la caché como consecuencia de un cambio del manifiesto.
4	UPDATEREADY	Una nueva versión de la caché está disponible. Hay un evento updateready correspondiente que se ejecuta en lugar del evento cached cuando una nueva actualización se ha descargado, pero todavía no se ha activado con el método swapCache() (ver más adelante).
5	OBSOLETE	El grupo cacheado está obsoleto.

2. Los eventos

Algunos eventos se producen según lo que se pasa con AppCache.

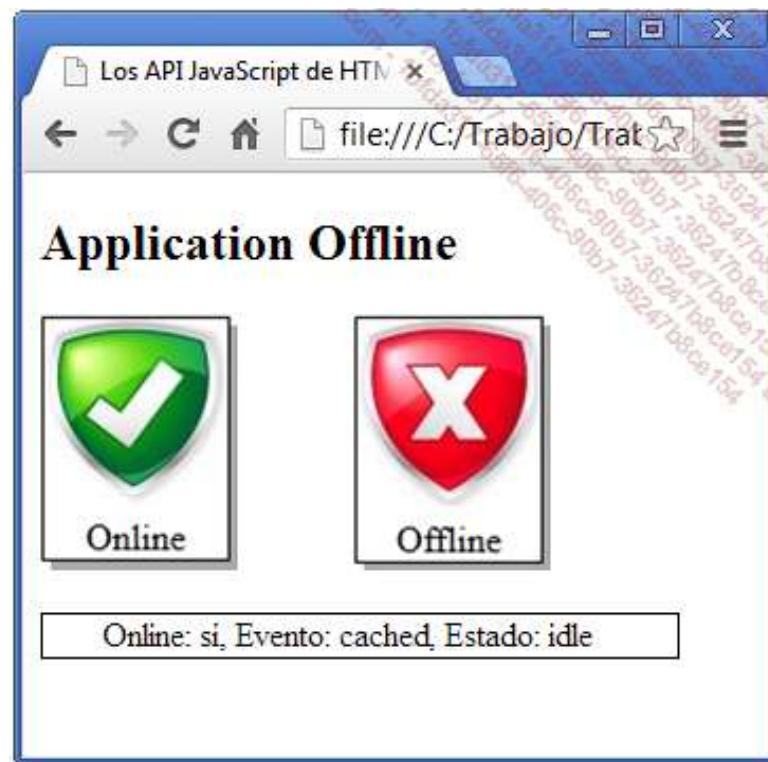
checking	Se produce cuando el navegador busca descargar el manifiesto la primera vez o verifica si hay disponible una actualización del manifiesto.
noupdate	Se produce si no hay versión de actualización del manifiesto disponible en el servidor.
downloading	Se produce cuando el navegador descarga el manifiesto la primera vez o una actualización de este.
progress	Se produce cuando cada archivo mencionado en el manifiesto se descarga en la caché.
cached	Se produce cuando todos los recursos se han descargado y están en la caché.
updateready	Se produce cuando todos los archivos mencionados en una actualización del manifiesto se han descargado o recargado. En ese caso, se puede usar el método swapCache() para que el navegador pueda utilizar estos archivos.
obsolete	Se produce cuando el archivo manifiesto no se puede encontrar (error 404 o 401).
error	Se desencadena cuando, por ejemplo, el archivo de manifiesto no se ha podido encontrar o existe un problema durante la descarga de un archivo o recurso.

En algunas ocasiones, estos eventos están precedidos del prefijo on: onchecking, onnoupdate, ondownloading, onprogress, etc.

Observe que, en el momento de escribir este libro, los navegadores no soportan de la misma manera los eventos de caché. Así, por ejemplo, Firefox ignora los eventos como checking y updateready, pero reconoce noupdate y error.

3. Application

Usando un nuevo ejemplo, vamos a ilustrar estas propiedades.



El script relativo a los estados y eventos devuelve Evento: cached. Todos los recursos señalados en el manifiesto se han descargado y están presentes en la caché. Con Estado: idle, indica que todos los recursos se han descargado y que no hay actualización en curso de la caché.

Las imágenes y los archivos de los ejemplos están disponibles en el página Información.

El archivo de base (index.html)

```
<!DOCTYPE html>
<html lang="en" manifest="offline.appcache">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<link rel="stylesheet" href="style.css">
<script src="script.js"></script>
</head>
<body onload="init()">
<h2>Application Offline</h2>


<div id="casilla">&nbsp;</div>
</body>
</html>
```

El script externo (script.js)

```
function init(){
var cacheStatusValues = [];
cacheStatusValues[0] = 'uncached';
cacheStatusValues[1] = 'idle';
cacheStatusValues[2] = 'checking';
cacheStatusValues[3] = 'downloading';
cacheStatusValues[4] = 'updateready';
cacheStatusValues[5] = 'obsolete';
```

```

var cache = window.applicationCache;
cache.addEventListener('cached' , logEvent, false);
cache.addEventListener('checking', logEvent, false);
cache.addEventListener('downloading', logEvent, false);
cache.addEventListener('error', logEvent, false);
cache.addEventListener('noupdate', logEvent, false);
cache.addEventListener('obsolete', logEvent, false);
cache.addEventListener('progress', logEvent, false);
cache.addEventListener('updateready', logEvent, false);
function logEvent(e) {
var message, online, status, type;
online = (navigator.onLine)? 'sí': 'no';
status = cacheStatusValues[cache.status];
type = e.type;
message = 'Online: ' + online;
message+= ', Evento: ' + type;
message+= ', Estado: ' + status;
document.querySelector('#casilla').innerHTML = message;
}
}

```

Comentario

```

var cacheStatusValues = [];
cacheStatusValues[0] = 'uncached' ;
cacheStatusValues[1] = 'idle';
...

```

Con la variable Array cacheStatusValues, cada valor de estado se asocia con su derecho.

```
var cache = window.applicationCache;
```

La variable cache recupera toda la información del objeto applicationCache.

```

cache.addEventListener('cached', logEvent, false);
...

```

Un manejador de eventos DOM 2 «escucha» el evento cached y, si se produce, pasa el control a la función logEvent (). Es lo mismo para el resto de los eventos.

```
online = (navigator.onLine)? 'sí': 'no';
```

La variable online comprueba (navigator.onLine) si la página está en línea o no.

```
type = e.type;
```

La variable tipo captura los eventos.

```
status = cacheStatusValues[cache.status];
```

cacheStatusValues [cache.status] proporciona el estado.

Lo que continúa es JavaScript clásico.

La hoja de estilo externa (style.css)

```

#casilla { width: 300px;
    border: 1px solid black;
    text-align: center;
    margin-top: 15px;}
```

El archivo de manifiesto

CACHE MANIFEST

CACHE:

index.html

style.css

script.js

on.png

off.png

FALLBACK:

NETWORK:

Actualización de los datos cacheados

1. El método update

El método `update()` verifica si existe una nueva versión del archivo del manifiesto. Si existe, inicia la descarga de la nueva versión.

El navegador verifica la existencia de un *timestamp* más reciente del archivo de manifiesto, así como una modificación del contenido de este.

Si un navegador descarga un archivo de manifiesto actualizado y comprueba que el contenido no ha cambiado, detiene el proceso de descarga y guarda los archivos cacheados anteriormente para ahorrar el ancho de banda de red.

En fase de aprendizaje o desarrollo, hay un truco que consiste en cambiar el número de versión (por ejemplo 1.0.1. en lugar de 1.0) o añadir una línea de comentario.

Aunque el navegador verifica automáticamente la existencia de un nuevo archivo de manifiesto, es posible forzar este proceso de actualización, si piensa que el archivo de manifiesto ha cambiado desde la primera visita del usuario, añadiendo al script:

```
setInterval(function () {window.applicationCache.update();}, 3600000);
```

Esta parte del script verifica cada 60 minutos si existe una nueva versión del manifiesto. Si es el caso, se descargan los archivos y recursos definidos en el nuevo manifiesto.

2. El método swapCache

El método `swapCache()` indica al navegador que comience a utilizar el nuevo contenido cacheado si se ha descargado una actualización. Es importante observar que, incluso si hay una nueva versión del manifiesto, la aplicación continuará usando los antiguos archivos de la caché (como se especifica en la antigua versión del manifiesto). Cuando se llama a la función `swapCache()`, entonces solo se usan los nuevos archivos de la caché.

Ejemplo de código que se puede utilizar

```
<!DOCTYPE html>
<html lang="en" manifest="demo.appcache">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<script type="text/javascript">
window.applicationCache.addEventListener('updateready',
function(){
window.applicationCache.swapCache();
}, false);
</script>
</head>
<body>
...
</body>
</html>
```

Presentación y objetivos

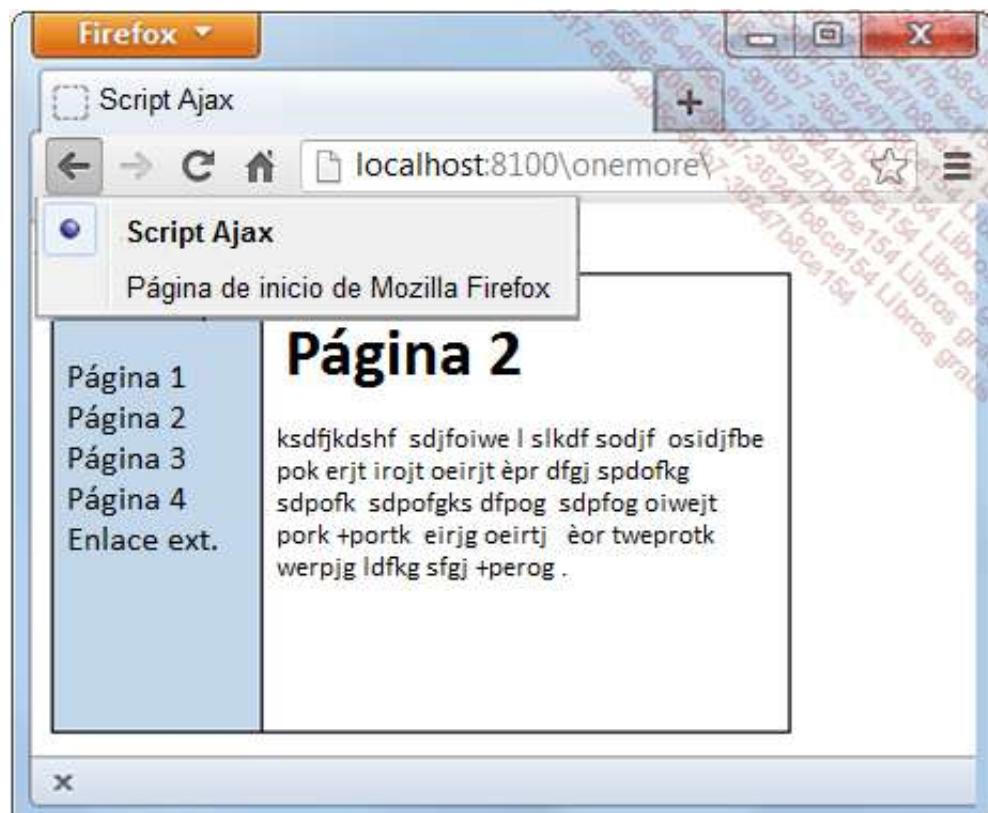
El API Html5 History permite manipular el histórico del navegador usando JavaScript. Ahora es posible cambiar dinámicamente la URL en la barra de direcciones del navegador y añadir esta nueva url en el histórico, sin recargar la página.

Hay que admitir que, desde que apareció la ola Ajax, el uso del histórico se ha visto perjudicado y el contenido actual de la página no está relacionado con su URL. Este API permitirá principalmente a los desarrolladores remediar estos inconvenientes bien conocidos de Ajax.

Ilustramos estos inconvenientes con algunas capturas de pantalla.



Independientemente del contenido que se muestra, este no aparece en el histórico.



El API History con el objeto `window.history` ofrece diferentes métodos:

`history.back()`

Para volver atrás en el histórico, es decir, volver a la página anterior.

`history.forward()`

Para avanzar en el histórico, es decir, ir a la página siguiente.

`history.pushState()`

Añadir entradas en el histórico del navegador.

onpopstate()	Interceptar los eventos relacionados con las modificaciones del histórico del navegador. Esto se hace con las funcionalidades Anterior y Siguiente.
history.replaceState()	Sustituye una entrada en el histórico del navegador.

Los métodos `window.history.back()` y `window.history.forward()` no necesitan explicaciones porque ya existen en JavaScript clásico.

y

El resto de los métodos se tratarán detalladamente a lo largo de este capítulo.

Observe que W3C informa que este API todavía está en una fase avanzada del desarrollo (<http://www.w3.org/TR/html5/browsers.html#history>).

Este API necesita pasar por un servidor, local o en línea.

Disponibilidad del API

El API está soportado por los navegadores de escritorio siguientes:

- Internet Explorer 10+
- Safari 5+
- Firefox 4+
- Chrome 8+
- Opera 11.5+

Estos son los navegadores de Smartphone y otras tabletas que soportan el API History:

- iOS Safari 4.2+
- Opera Móvil 11.1+
- Android 2.2+

Para probar la disponibilidad del API, el código que se utiliza con más frecuencia es:

```
function support_history_api() {  
    return !(window.history && history.pushState);  
}
```

de una manera más concisa pero menos completa:

```
if (typeof history.pushState !== "undefined") {  
...  
}
```

Ejemplo

```
<!DOCTYPE html>  
<html lang="es">  
<head>  
<title>Los API JavaScript de Html5</title>  
<meta charset=utf-8>  
<style type="text/css">  
#box { width:400px;  
        border: 1px solid black;  
        background-color: rgb(195,215,235);  
        text-align: center; }  
</style>  
<script type="text/javascript">  
var support = false;  
function isSupportedHistory() {  
return !(window.history && history.pushState);  
}  
function init() {  
support = isSupportedHistory();  
if(support) {  
msg = "El API History está soportado";  
document.querySelector('#box').innerHTML = msg;  
}  
else {  
msg = "El API History no está soportado";  
document.querySelector('#box').innerHTML = msg;  
}  
}  
window.onload = function() {  
init();
```

```
    }
</script>
</head>
<body>
<h2>Browser History</h2>
<div id="box">&nbsp;</div>
</div>
</body>
</html>
```

El resultado en Safari 5:



Introducir nuevas entradas en el histórico

Para introducir una nueva entrada en el histórico, el API History ofrece el método `pushState()`. La sintaxis es:

```
history.pushState(estado, título, url)
```

Este método tiene tres argumentos:

`estado` Un objeto JavaScript o una estructura JSON para la nueva entrada en el histórico.

`título` El título de la nueva entrada. Es importante observar (sobre todo para los ejemplos) que, actualmente, este argumento solo está implantado en alguno de los navegadores más recientes.

`url` La URL (relativa o absoluta) de la nueva entrada en el histórico.

Es importante observar que, si se transmite una URL, el navegador cambiará la URL mostrada en su barra de direcciones. La nueva URL debe pertenecer al mismo dominio, pero el resto se puede modificar.

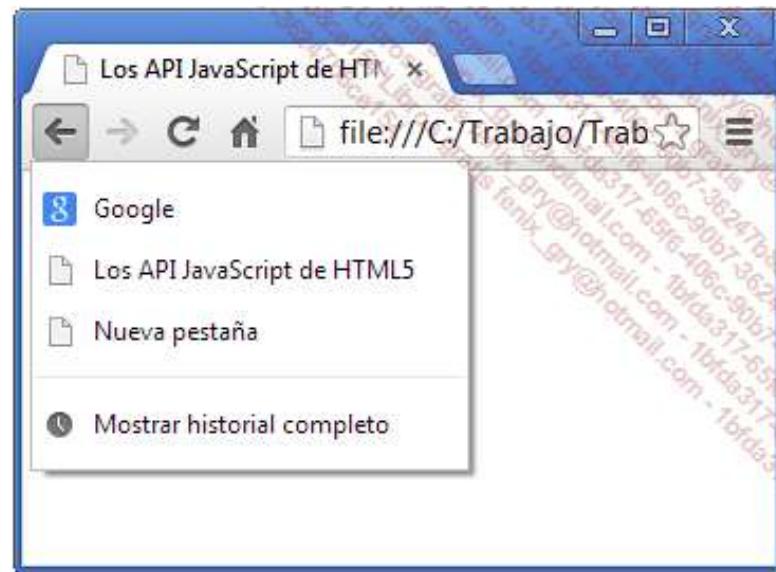
Ejemplo

Al hacer clic en un botón, añadimos entradas en el histórico del navegador.

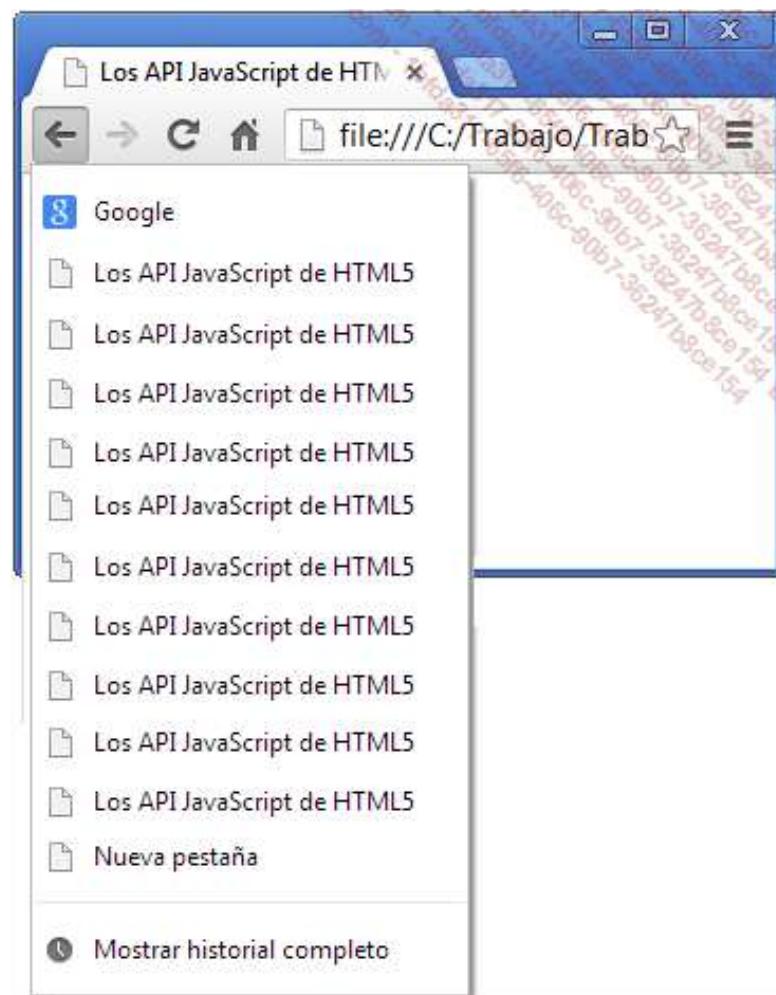
La situación inicial:



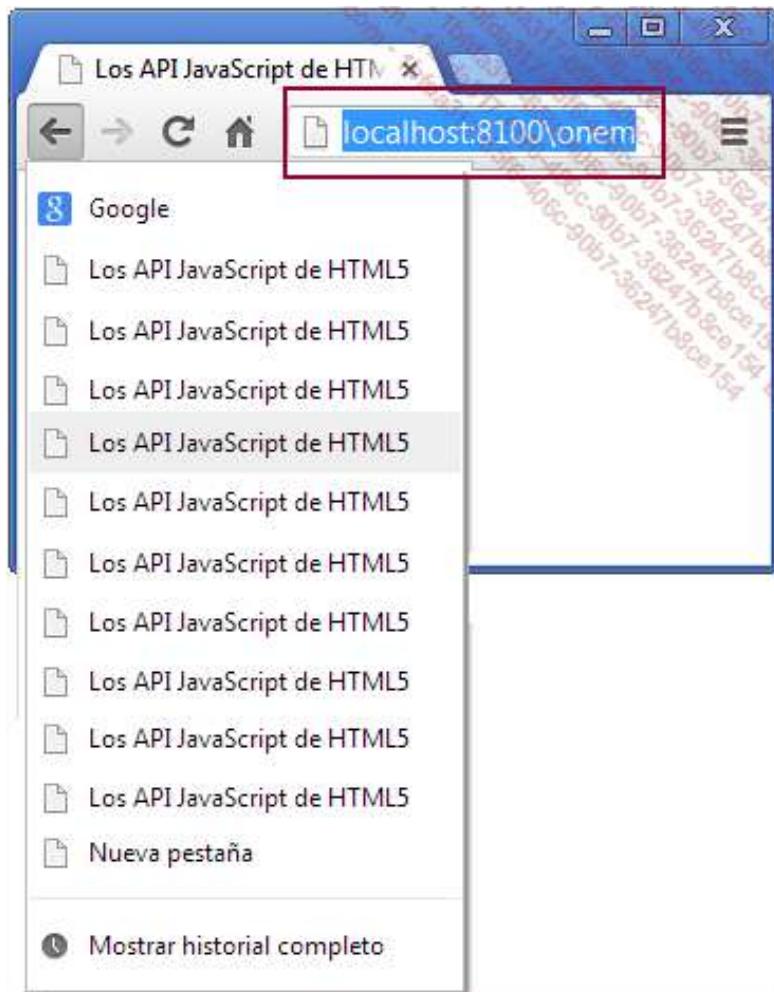
Echamos un vistazo al histórico (clic con el botón derecho del ratón en el botón **Anterior**).



Después de haber pulsado en el botón **Go**, el histórico (clic con el botón derecho del ratón) se convierte en:

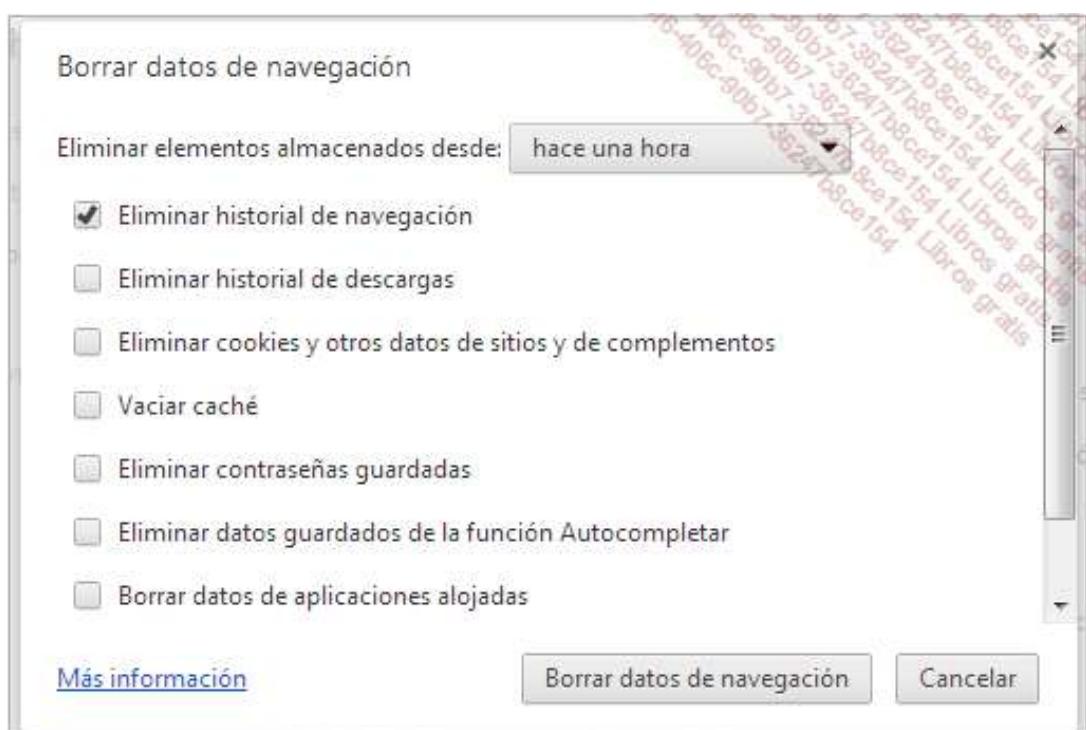


Y si pulsamos en un elemento del histórico, aparece su URL en la barra de direcciones.

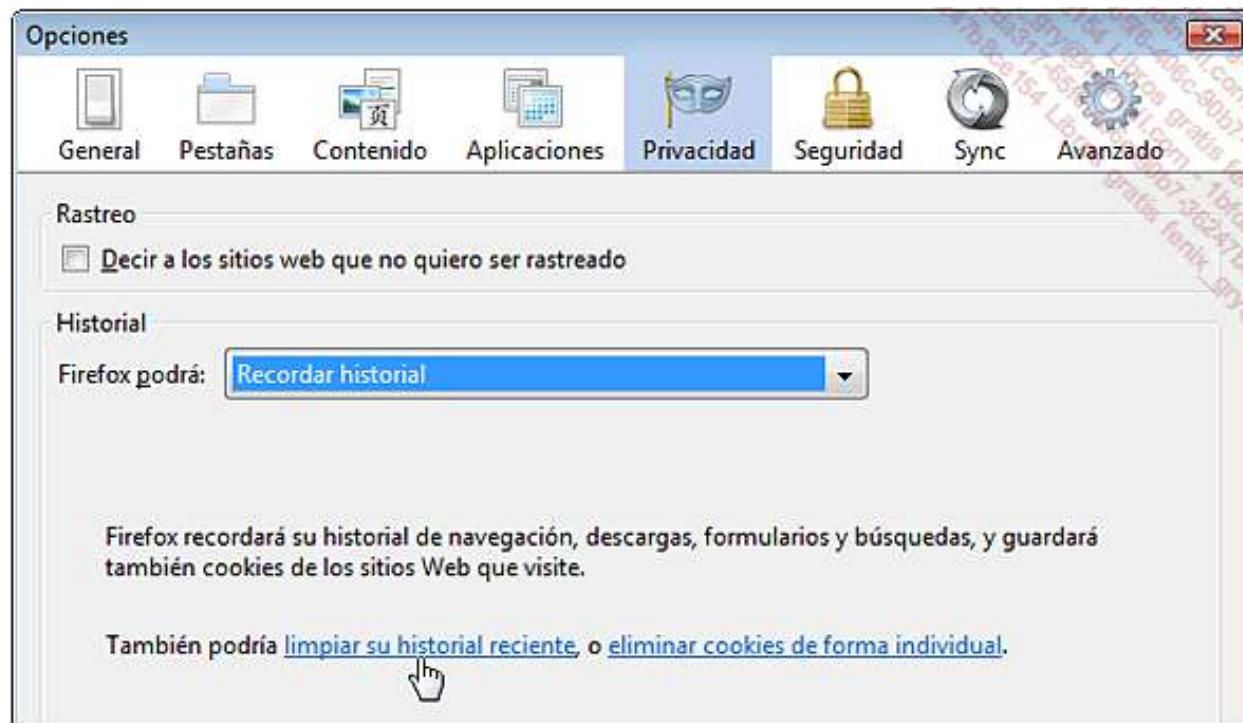


► Durante las pruebas y desarrollos, será necesario borrar a menudo el histórico.

El procedimiento para Google Chrome es Herramientas - **Borrar datos de navegación**.



Para Firefox, en la barra de menús: **Herramientas - Opciones** - pestaña **Privacidad** - **Historial** - enlace **limpiar su historial reciente**.



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
button { margin-left: 60px; }
</style>
<script type="text/javascript">
var support = false;
function isSupportedHistory() {
return !(window.history && history.pushState);
}
function init() {
support = isSupportedHistory();
if(support) {
for( var i = 0; i < 10; i++ ) {
history.pushState(i, 'i is ' + i, '/' + i + '.html ');
}
}
}
</script>
</head>
<body>
<h2>El API History</h2>
<p>Añadir entradas al histórico</p>
<button onclick="init()">Go</button>
</body>
</html>
```

Comentario

```
var support = false;
function isSupportedHistory() {
```

```
return !(window.history && history.pushState);  
}  
function init() {  
support = isSupportedHistory();
```

Este código verifica si el navegador soporta el API.

```
for( var i = 0; i < 10; i++ ) {  
history.pushState(i, 'i is ' + i, '/' + i + '.html );  
}
```

Las diferentes entradas se crean (`history.pushState`) con un sencillo bucle `for`.

Los eventos relacionados con los cambios en el histórico

El método `window.onpopstate()` intercepta los eventos relacionados con las modificaciones del histórico del navegador. Estas se producen con las funcionalidades **Anterior** y **Siguiente**. La propiedad `state` del objeto evento contiene el objeto transmitido.

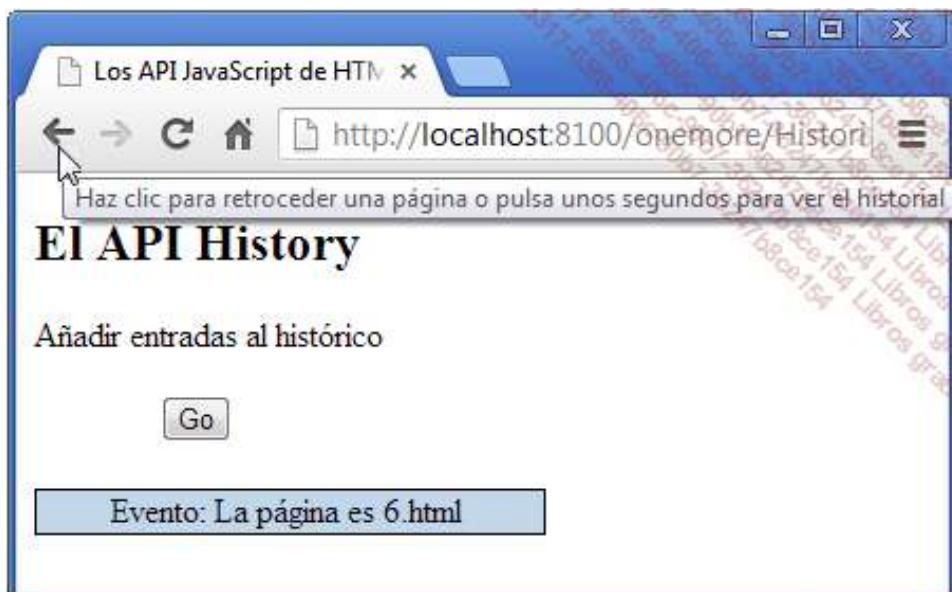
Ejemplo 1

Retomemos el ejemplo anterior y añadamos el método `window.onpopstate()` para ver los eventos.

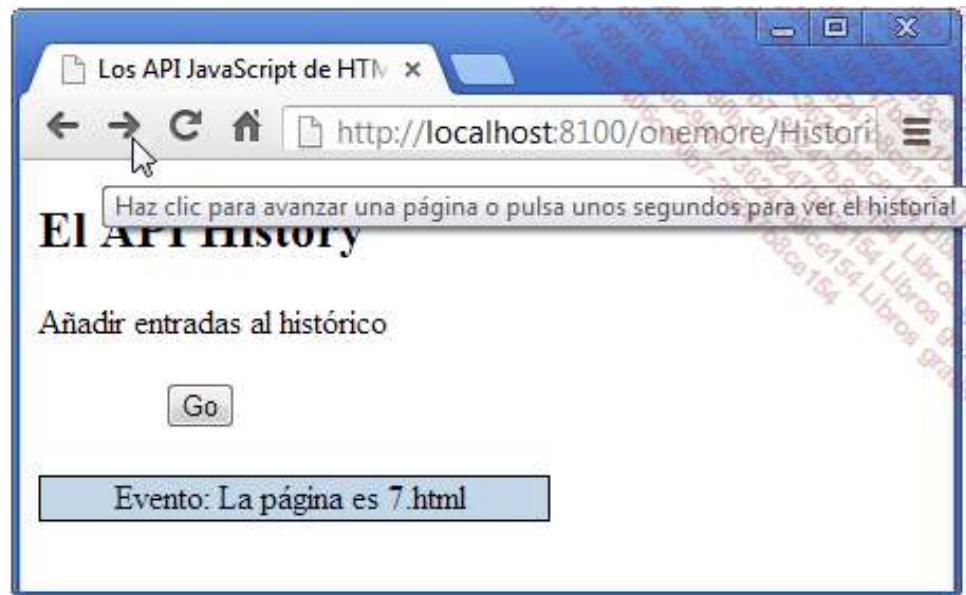
Cuando se abre la página:



Al hacer clic en el botón **Anterior**, se desencadena un evento que es interceptado por `onpopstate()` y se muestra en la capa gris.



Ídem para el botón **Siguiente**.



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#box { width: 240px;
        border: 1px solid black;
        background-color: rgb(195,215,235);
        margin-top: 20px;
        text-align: center;}
button { margin-left: 60px;}
</style>
<script type="text/javascript">
var support = false;
function isSupportedHistory() {
return !(window.history && history.pushState);
}
function visualizarPopState(event) {
if(event.state != null) {
msg = ";Evento! La página es " + event.state + ".html";
document.querySelector('#box').innerHTML = msg;
}
}
function init() {
support = isSupportedHistory();
if(support) {
for( var i = 0; i < 10; i++ ) {
history.pushState(i, 'i is ' + i, '/' + i + '.html ');
}
window.onpopstate = visualizarPopState;
}
}
</script>
</head>
<body>
<h2>El API History</h2>
<p>Añadir entradas al histórico</p>
<button onclick="init()">Go</button>
<div id="box">&nbsp;</div>
</body>
</html>
```

Comentario

El inicio del script es idéntico al anterior.

```
function visualizarPopState(event) {  
if(event.state != null) {  
msg = "¡Evento!, el estado es: " + event.state;  
document.querySelector('#box').innerHTML = msg;  
}  
}
```

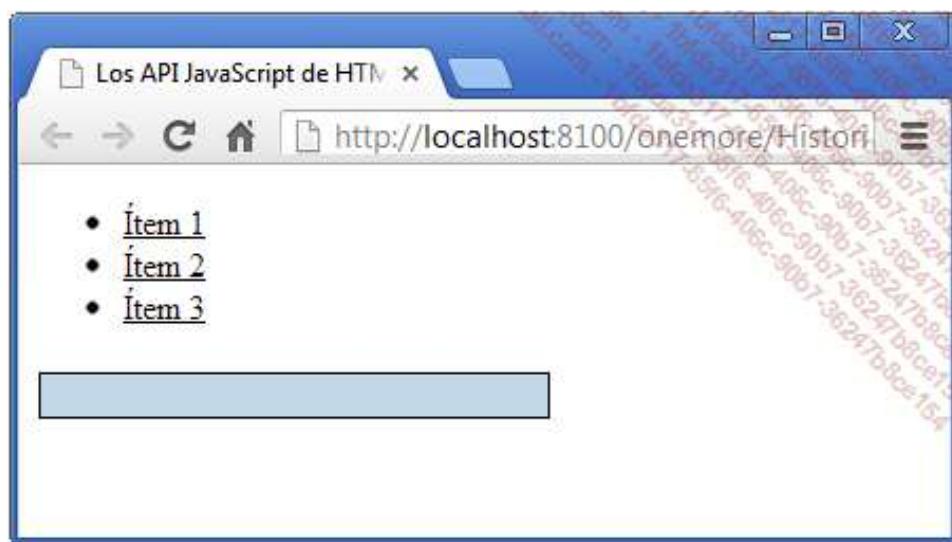
Si se produce un evento (`event.state != null`), este se muestra en la capa box.

Ejemplo 2

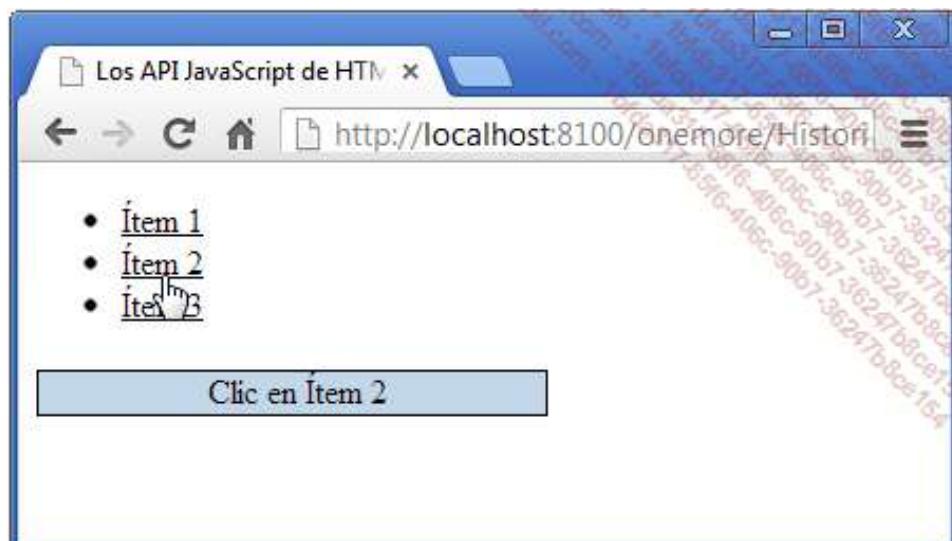
Consideremos los enlaces que se muestran en una lista con viñetas. Recogemos los diferentes eventos relacionados después de aplicar el método `pushState`.

Este script llama a jQuery porque este framework simplifica mucho la búsqueda (`query`) de la página y del DOM.

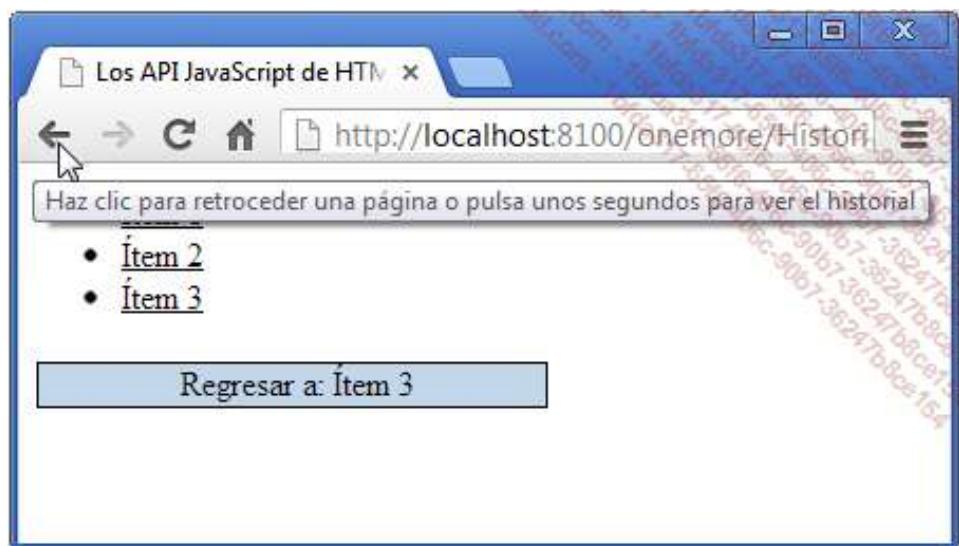
Situación inicial:



Al hacer clic en un ítem:



Usando la función **Anterior**:



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#box { width: 240px;
        border: 1px solid black;
        background-color: rgb(195,215,235);
        margin-top: 20px;
        text-align: center;}
a { color: black;}
</style>
<script type="text/javascript" src="http://ajax.googleapis.com/
ajax/libs/jquery/1.7/jquery.min.js">
</script>
<script type="text/javascript">
$(document).ready(function() {
window.onpopstate = function(event) {
if (event.state) {
message = 'Regresar a: ' + event.state.name;
document.querySelector('#box').innerHTML = message;
}
};
$('a').click(function(event) {
var value = $(this).attr('class');
window.history.pushState({ name: value }, '', this.href);
message = 'Clic en ' + value;
document.querySelector('#box').innerHTML = message;
});
});
</script>
</head>
<body>
<ul>
<li><a href="#Item 1" class="Item 1">Ítem 1</a></li>
<li><a href="#Item 2" class="Item 2">Ítem 2</a></li>
<li><a href="#Item 3" class="Item 3">Ítem 3</a></li>
</ul>
<div id="box">&nbsp;</div>
</body>
</html>
```

Comentario

```
$(document).ready(function() {  
...  
});
```

Instrucción clásica de jQuery que no desencadena el script hasta que el DOM esté totalmente disponible (ready).

```
window.onpopstate = function(event) {  
if (event.state) {  
message = 'Volver a: ' + event.state.name;  
document.querySelector('#box').innerHTML = message;  
}  
};
```

Intercepción del evento generado por el clic en **Anterior** (event.state) y su visualización en la capa box.

```
$('.a').click(function(event) {  
var value = $(this).attr('class');
```

En jQuery, al hacer clic en uno de los enlaces (`$('.a')`), la variable `value` contendrá el valor del atributo de su clase (class).

```
window.history.pushState({ name: value }, '', this.href);
```

Se añadirá (pushState) esta variable `value`, que contiene el valor del atributo `class`.

```
message = 'Clic en ' + value;  
document.querySelector('#box').innerHTML = message;  
});  
});
```

El valor de la variable `value` se carga en la capa box.

Modificar una entrada del histórico

Para modificar una entrada en el histórico, el API History ofrece el método `replaceState()`. La sintaxis es:

```
history.replaceState(estado, título, url)
```

Este método tiene tres argumentos:

`estado` Un objeto JavaScript o una estructura JSON para la nueva entrada en el histórico.

`título` El título de la nueva entrada. Observe (sobre todo en los ejemplos) que, a día de hoy, este argumento solo está implantado en alguno de los navegadores más recientes.

`url` La URL (relativa o absoluta) de la nueva entrada en el histórico.

Gracias a este método `replaceState`, la página inicial se podrá revisitar después de que se haya introducido el nuevo histórico usando `pushState`.

Ejemplo

Ilustremos este método `replaceState` después de haber introducido dos nuevas entradas en el histórico.

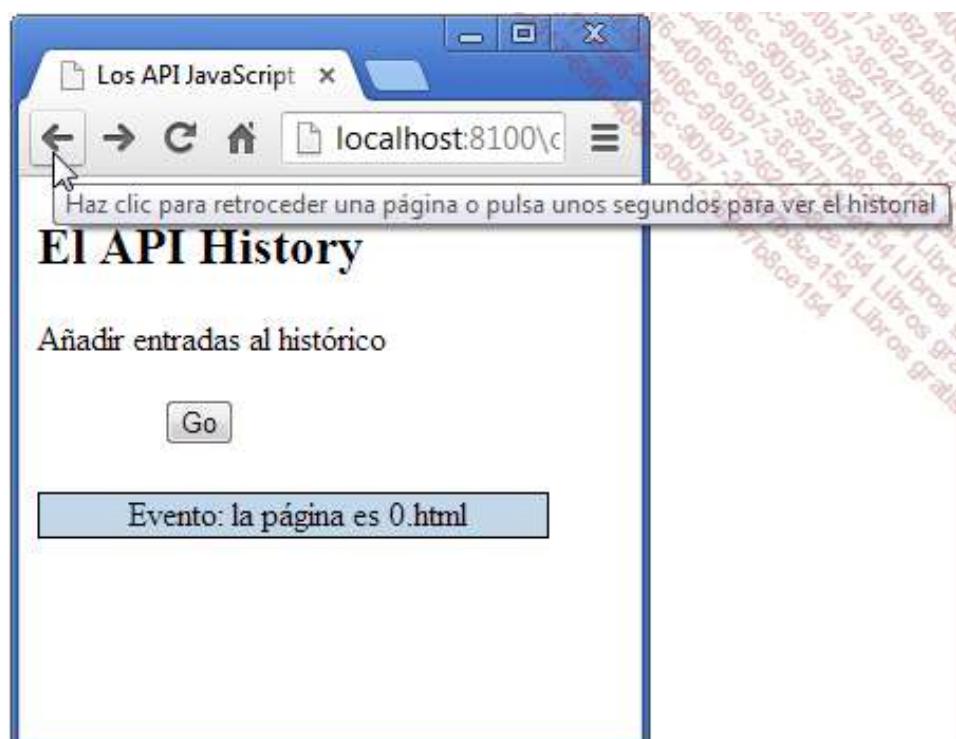
La página inicial:



Después de haber pulsado en el botón **Go**, las dos entradas se han creado (ver `localhost/2.htm`).



Después de haber pulsado dos veces en el botón **Anterior**, volvemos a la página inicial a pesar de haber añadido dos entradas en el histórico. Observe que el método `replaceState()` renombra la página a `0.html`.



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#box { width: 240px;
       border: 1px solid black;
```

```

background-color: #c5e1f5;
margin-top: 20px;
text-align: center;}
button { margin-left: 60px; }
</style>
<script type="text/javascript">
var support = false;
function isSupportedHistory() {
return !(window.history && history.pushState);
}
function visualizarPopState(event) {
if(event.state != null) {
msg = "Evento: la página es " + event.state + ".html";
document.querySelector('#box').innerHTML = msg;
}
}
function init() {
support = isSupportedHistory();
if(support) {
history.replaceState(0, 'el número 0', null);
history.pushState(1, 'el número 1', '1.html ');
history.pushState(2, 'el número 2', '2.html ');
window.onpopstate = visualizarPopState;
}
}
</script>
</head>
<body>
<h2>El API History</h2>
<p>Añadir entradas al histórico</p>
<button onclick="init()">Go</button>
<div id="box">&nbsp;</div>
</body>
</html>

```

Comentario

```
history.replaceState(0, 'el número 0', null);
```

El método `replaceState` asigna a la página actual (`localhost/replace State.htm`) la posición 0 en el histórico.

```
history.pushState(1, 'el número 1', '1.html ');
history.pushState(2, 'el número 2', '2.html ');
```

Creación de dos entradas en el histórico que toman las posiciones 1 y 2.

```
window.onpopstate = visualizarPopState;
```

El método `onpopState` llama a la función `visualizarPopState` que, como habrá adivinado, muestra los eventos en la capa `box`.

Aplicación final

Los diaporamas o galerías de fotos forman parte de las aplicaciones que afectan negativamente al histórico. Normalmente es imposible meter en los favoritos una imagen en particular. Con el API History, a partir de ahora es posible inscribir cada elemento en el histórico.

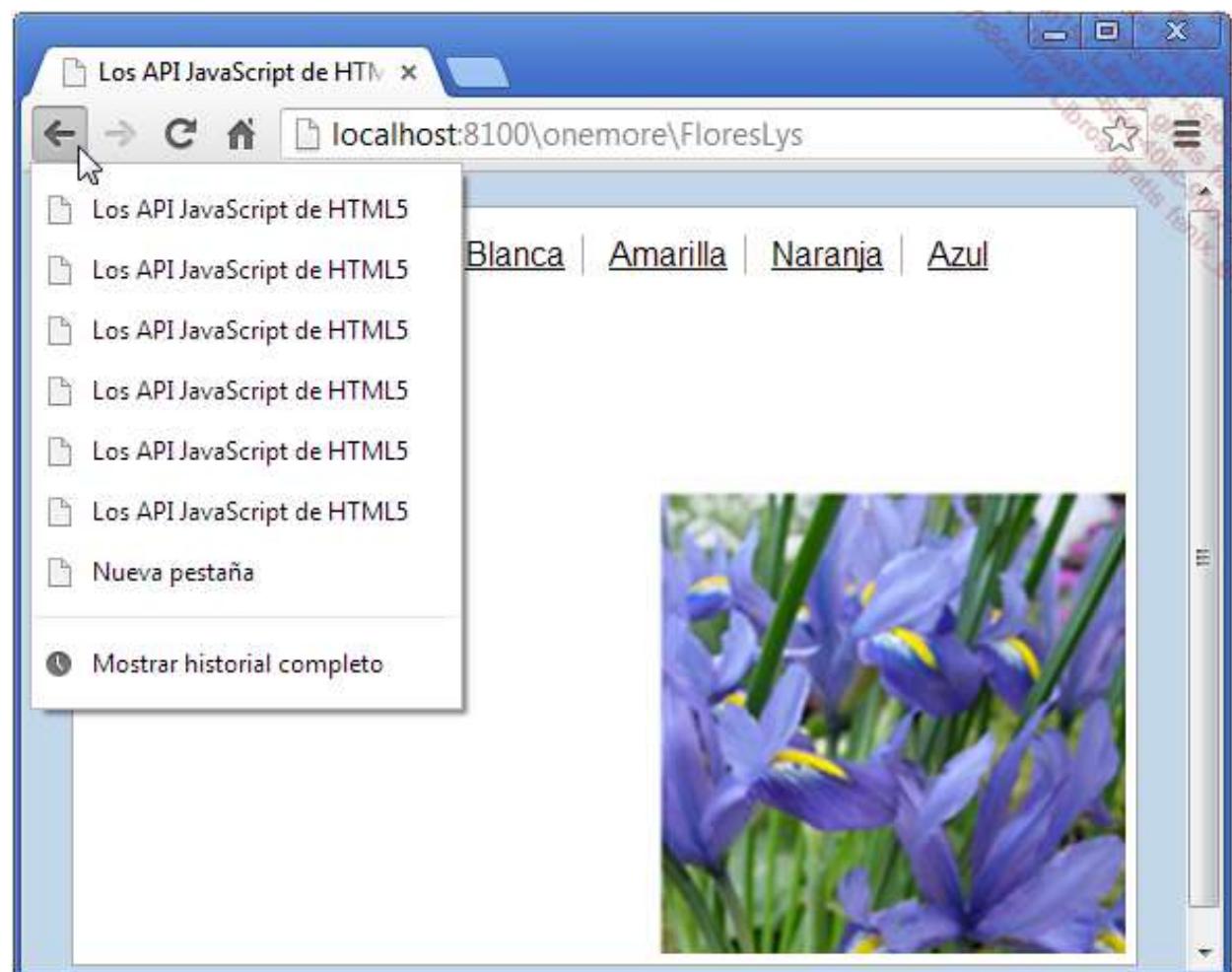
La página de bienvenida de presentación del diaporama:



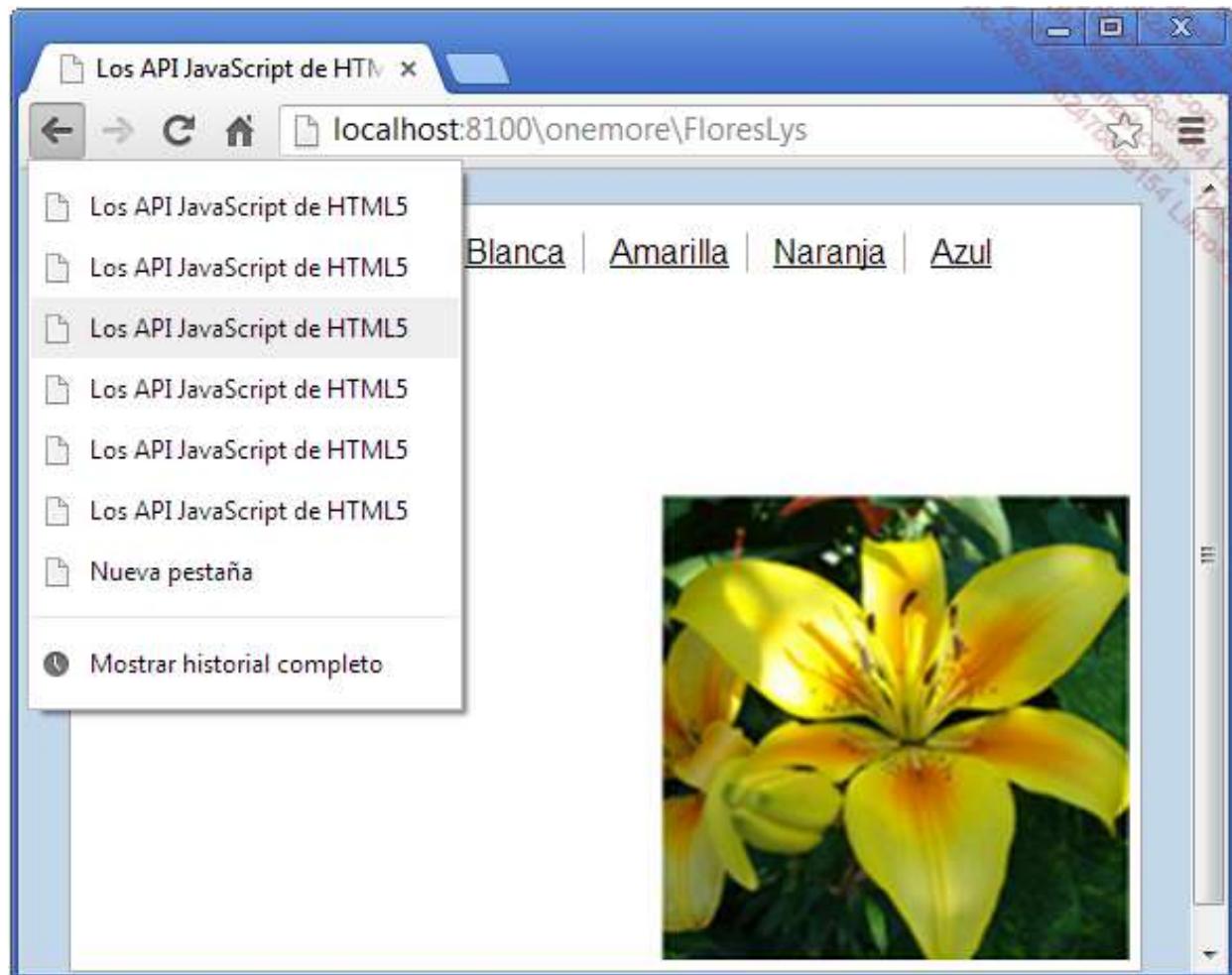
Visitamos los diferentes enlaces propuestos.



Un vistazo al histórico. Por cada enlace visitado, se ha creado una entrada en el histórico.



Y este nuevo histórico es totalmente manejable.



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
html { background-color: rgb(195,215,235); }
body { margin: 1em auto;
      width: 400px;
      background-color: white;
      border: solid 1px #aaa;
      padding: 5px;
      font-family: Arial,sans-serif; }
h1 { font-family: Arial,sans-serif;
     float: left;
     width: 30%;
     margin: 0; }
nav { display: block;
      float: right;
      width: 65%; }
ul { list-style: none;
     padding: 0;
     margin: 0; }
li { display: inline-block;
     padding: 0;
     border-right: solid 1px #aaa;
     margin:.5em 0 0; }
li:last-child { border-right:0; }
a { color: black;
```

```
    padding:.2em 0.5em; }
a:hover { text-decoration:none; }
#content { clear:left;
            float:left;
            width:45%;
            margin-right:10%;
            line-height:1.4em; }
#foto { float:right;
            width:45%;
            margin-top:1em; }
.cf:before, .cf:after { content:""; display:table; }
.cf:after { clear:both; }
</style>
</head>
<body class="cf">
<h1>Las flores de Lys</h1>
<nav>
<ul>
<li><a href="/demo/history/enlace">Blanca</a></li>
<li><a href="/demo/history/amarillo">Amarilla</a></li>
<li><a href="/demo/history/naranja">Naranja</a></li>
<li><a href="/demo/history/azul">Azul</a></li>
</ul>
</nav>
<p id="content">Inicio</p>

<script type="text/javascript">
var contenido = document.getElementById('content'),
fotos = document.getElementById('foto'),
enlaces = document.getElementsByTagName('a'),
lys = {
blanco: {
content: 'Lys blanco',
foto: 'http://www.lehtml.com/imagenes/blanco.png'
},
amarillo: {
content: 'Lys amarilla',
foto: 'http://www.lehtml.com/imagenes/amarillo.png'
},
naranja: {
content: 'Lys naranja',
foto: 'http://www.lehtml.com/imagenes/naranja.png'
},
azul: {
content: 'Lys azul',
foto: 'http://www.lehtml.com/imagenes/azul.png'
}
};
function updateContent(data) {
if (data == null)
return;
contenido.textContent = data.content;
fotos.src = data.foto;
}
function clickHandler(event) {
var flor = event.target.getAttribute('href').split('/').pop(),
data = lys[flor] || null;
updateContent(data);
history.pushState(data, event.target.textContent,
event.target.href);
return event.preventDefault();
}
for (var i = 0, l = enlaces.length; i < l; i++) {
enlaces[i].addEventListener('click', clickHandler, true);
}
```

```
window.addEventListener('popstate', function(event) {  
updateContent(event.state);  
});  
history.replaceState({  
content: contenido.textContent,  
foto: fotos.src  
, document.title, document.location.href);  
</script>  
</body>  
</html>
```

Comentario

```
function clickHandler(event) {  
var flor = event.target.getAttribute('href').split('/').pop(),  
data = lys[flor] || null;  
updateContent(data);
```

Cargar los datos JSON en la página (ver lys = {...}).

```
history.pushState(data, event.target.textContent, event.target.href);  
return event.preventDefault();  
}
```

Añade una entrada (history.pushState) al histórico.

```
for (var i = 0, l = enlaces.length; i < l; i++) {  
enlaces[i].addEventListener('click', clickHandler, true);  
}
```

Añade un manejador de eventos que, al hacer clic en un enlace, llama a la función clickHandler().

```
window.addEventListener('popstate', function(event) {  
updateContent(event.state);  
});
```

Gracias a la información de popstate, permite regresar a una página anterior.

```
history.replaceState({  
content: contenido.textContent,  
foto: fotos.src  
, document.title, document.location.href);
```

Guarda la página de inicio de manera que podamos volver a visitarla incluso si falla la adición al histórico.

Presentación, objetivos y precauciones

La funcionalidad de drag&drop permite al usuario mover elementos de la página, manteniendo el botón del ratón pulsado, y depositarlos en otro lugar soltando el botón del ratón.

Este drag&drop permite una interacción con el usuario adornada con un efecto gráfico agradable. Y no olvidemos que, con nuestros móviles y tabletas táctiles, el desplazamiento de elementos en la pantalla es un comportamiento muy normal.

Este API Drag and Drop (DnD) ofrece de manera nativa el drag&drop, funcionalidad que hizo que muchos frameworks, como jQuery, Dojo, MooTools y otros, tuvieran tanto éxito.

Según el estado de avance de este API a día de hoy, el drag&drop de Html5 está lejos de ser perfecto y recibe numerosas críticas. De hecho, hay muchos comportamientos extraños y otras inconsistencias entre las interpretaciones que hacen los navegadores, incluso los más modernos (Firefox, Chrome y Safari). De esta manera es muy difícil, incluso imposible, hacer un script de drag&drop totalmente compatible. Una pena para una API que es nativo.

A continuación enumeramos algunas de las abundantes críticas:

- El API para el drag&drop es horrible (*The drag-and-drop API is horrible*).
- El desastre del drag&drop de Html5 (*The HTML5 drag and drop disaster*)http://www.quirksmode.org/blog/archives/2009/09/the_html5_drag.html
- Este módulo se debería retirar inmediatamente de la especificación Html5 (*The module should be removed from the HTML5 specification straight away*).
- Los navegadores deberían eliminar este API a la primera oportunidad y rehacerlo en profundidad (*Browsers should disable it at their earliest opportunity pending a complete rewrite from the ground up*).

A día de hoy, el mejor consejo que podemos dar a los desarrolladores es seguir utilizando el drag&drop de su framework JavaScript para hacer esta operación.

Este API Drag and Drop funciona en local.

Disponibilidad del API

El API Html5 Drag and Drop se basa en la implementación que hizo inicialmente Microsoft, disponible desde la versión 5.5 de Internet Explorer. Ahora está soportada por el resto de los navegadores.

- Internet Explorer 10+
- Firefox 3.6+
- Google Chrome 3.0+
- Safari 3.2+
- Opera 11.6+

Los Smartphone y otras tabletas utilizan una técnica propia.

Según la información disponible en la Red, la detección de este API no es fácil. El siguiente script usa el truco de crear una etiqueta y de probar si soporta el atributo draggable.

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#box { width: 300px;
       border: 1px solid black;
       background-color: rgb(195,215,235);
       text-align: center; }
</style>
<script type="text/javascript">
function init() {
if('draggable' in document.createElement('span')) {
msg = "El API Drag and Drop está soportado";
document.querySelector('#box').innerHTML = msg;
}
else {
msg = "El API Drag and Drop no está soportado";
document.querySelector('#box').innerHTML = msg;
}
}
window.onload = function() {
init();
}
</script>
</head>
<body>
<h2>Drag and Drop</h2>
<div id="box">&nbsp;</div>
</div>
</body>
</html>
```



Este script funciona bien con Firefox, Chrome, Safari, pero (y es una pena) no con Internet Explorer.

Para los amantes de la precisión, hay que utilizar la librería JavaScript Modernizr (<http://www.modernizr.com/download/>). Esta pequeña librería JavaScript Open Source bajo licencia MIT se ha especializado en detectar el soporte de las características HTML5 & CSS3.

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de HTML5</title>
<meta charset=utf-8>
<style type="text/css">
#box { width:300px;
       border: 1px solid black;
       background-color: rgb(195,215,235);
       text-align: center; }
</style>
<script type="text/javascript" src="Modernizr.js"></script>
<script type="text/javascript">
function init() {
if (Modernizr.draganddrop) {
msg = "El API Drag and Drop está soportado";
document.querySelector('#box').innerHTML = msg;
}
else {
msg = "El API Drag and Drop no está soportado";
document.querySelector('#box').innerHTML = msg;
}
}
window.onload = function() {
init();
}
</script>
</head>
<body>
<h2>Drag and Drop</h2>
<div id="box">&nbsp;</div>
</div>
</body>
</html>
```



Ahora la detección es totalmente compatible con todos los navegadores.

Definición del objeto desplazable

1. El atributo draggable

La primera etapa para hacer un elemento desplazable es añadir el atributo `draggable=true` al elemento que desea mover.

Ejemplo de capa desplazable:

```
<div id="box" draggable="true">
```

2. El objeto dataTransfer

El objeto `dataTransfer` es un nuevo objeto de Html5. Permite transferir datos entre diferentes eventos. Una vez que un dato está relacionado con un evento, este estará disponible (y recuperable) para el resto de los eventos. Esto se aplica perfectamente al drag&drop.

3. La propiedad effectAllowed

La propiedad `effectAllowed` permite definir el tipo de desplazamiento que el desarrollador desea o autoriza para el drag&drop.

Los valores son:

- `all`: el elemento se puede copiar, mover y unir.
- `copy`: el elemento se puede copiar.
- `copyLink`: el elemento se puede copiar y unir.
- `copyMove`: el elemento se puede copiar y mover.
- `link`: el elemento se puede unir.
- `linkMove`: el elemento se puede mover y unir.
- `move`: el elemento se puede mover.
- `none`: el elemento no se puede mover.
- `uninitialized`: el efecto no está inicializado.

Los valores por defecto son `move` para los elementos Html (como por ejemplo una capa), `link` para los enlaces y `copy` para las imágenes.

4. Los métodos setData y getData

Los métodos `setData()` y `getData()` permiten pasar datos entre el elemento desplazable y el destino. Para ello, en primer lugar el script del drag&drop debe usar el método `setData()` con objeto de relacionar un elemento con un evento (por ejemplo, el evento `ondragstart` - cf. sección siguiente). El destino, aquí la zona del drop, recuperará el elemento durante un evento que le es propio (por ejemplo, durante el evento `ondrop` - ver Los eventos del objeto desplazable (drag)) por el método `getData()`.

La sintaxis de `setData()` es:

```
setData(dataType, data)
```

donde dataType determina el tipo de datos. Hasta ahora, los dos tipos de datos aceptados son "Text" y "Url" y data define los datos, es decir, el elemento implicado por el drag&drop.

Ejemplo

```
ev.dataTransfer.setData("Text", ev.target.getAttribute('id'));
```

La sintaxis de getData() es:

```
getData(dataType)
```

donde datatype es el tipo de datos (Text o Url).

Ejemplo

```
ev.dataTransfer.getData("Text")
```

Los eventos del objeto desplazable (drag)

Muchos (quizás demasiados) eventos siguen el desplazamiento iniciado por el drag&drop. Comentamos alguno de ellos relacionados con el drag:

ondragstart Evento que se desencadena al inicio del desplazamiento, es decir, tan pronto como el usuario empieza a modificar la posición de un elemento.

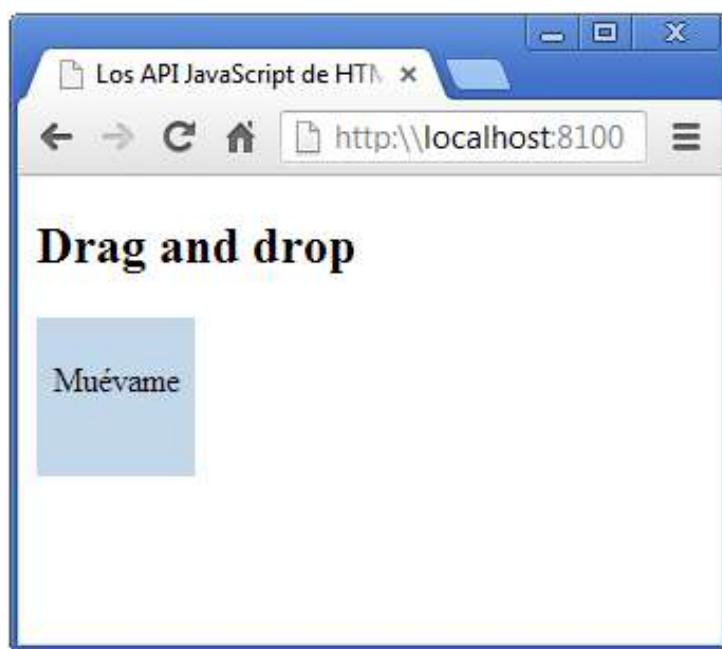
ondrag Evento que se produce cuando el elemento se mueve. En realidad, este evento se desencadena a intervalos regulares, cada 350 ms para ser precisos. Si el desplazamiento se para o anula, el valor que se devuelve es falso.

Los eventos relacionados con el drop se abordarán en la sección Definición de la zona del drop, de este capítulo.

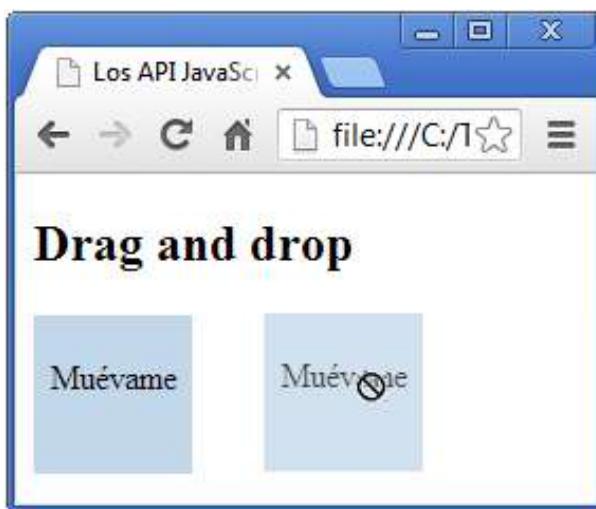
Un primer ejemplo de desplazamiento

Vamos a ilustrar un elemento desplazable. Este ejemplo es básico porque el efecto más espectacular es el drop.

Cuando se abre la página:



Durante el desplazamiento de la capa:



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#box { width:75px;
       height:75px;
       background-color: rgb(195,215,235);
       text-align: center; }
</style>
<script type="text/javascript">
```

```
function dragStart(ev) {  
ev.dataTransfer.effectAllowed='move';  
ev.dataTransfer.setData("Text", ev.target.getAttribute('id'));  
}  
</script>  
</head>  
<body>  
<h2>Drag and drop</h2>  
<div id="box" draggable="true" ondragstart="dragStart(ev)">  
<br>Muévame  
</div>  
</body>  
</html>
```

Comentario

```
<div id="box" draggable="true" ondragstart="dragStart(ev)">
```

El atributo `draggable="true"` hace que la capa sea desplazable. Al inicio del desplazamiento (`ondragstart`), se llama a la función `dragStart()`.

```
function dragStart(ev) {  
ev.dataTransfer.effectAllowed='move';  
ev.dataTransfer.setData("Text", ev.target.getAttribute('id'));  
}
```

Al evento `dragstart` que se pasa como argumento de la función (`ev`) asociamos la propiedad `move` al objeto `dataTransfer` (`dataTransfer.effectAllowed`). Los argumentos `setData` son `Text` y el elemento tiene un identificador (`getAttribute('id')`).

Los eventos del objeto desplazable (drop)

Los eventos relacionados con drop son:

ondragenter	Evento que se desencadena cuando el elemento que se mueve entra en otro elemento.
ondragleave	Evento que se desencadena cuando el elemento que se mueve sale de otro elemento.
ondragover	Evento que se ejecuta cuando el elemento desplazado pasa sobre un elemento. Si permite a este evento que continúe su propagación (no devolviendo el valor false), no será posible depositar el objeto que desplaza en este elemento.
ondrop	Evento que se desencadena cuando dejamos el elemento que se mueve, es decir, al soltar el botón del ratón.
ondragend	Evento que representa el final del desplazamiento ejecutado.

Definición de la zona del drop

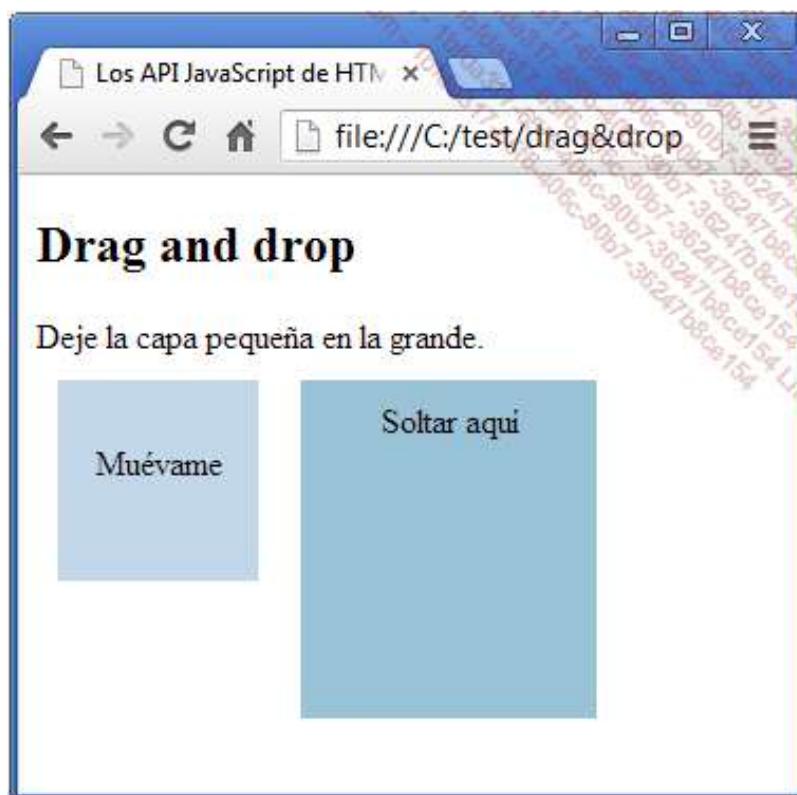
Para permitir que un elemento transportable se pueda soltar, es obligatorio anular el comportamiento por defecto del evento ondragover.

Ejemplo de código

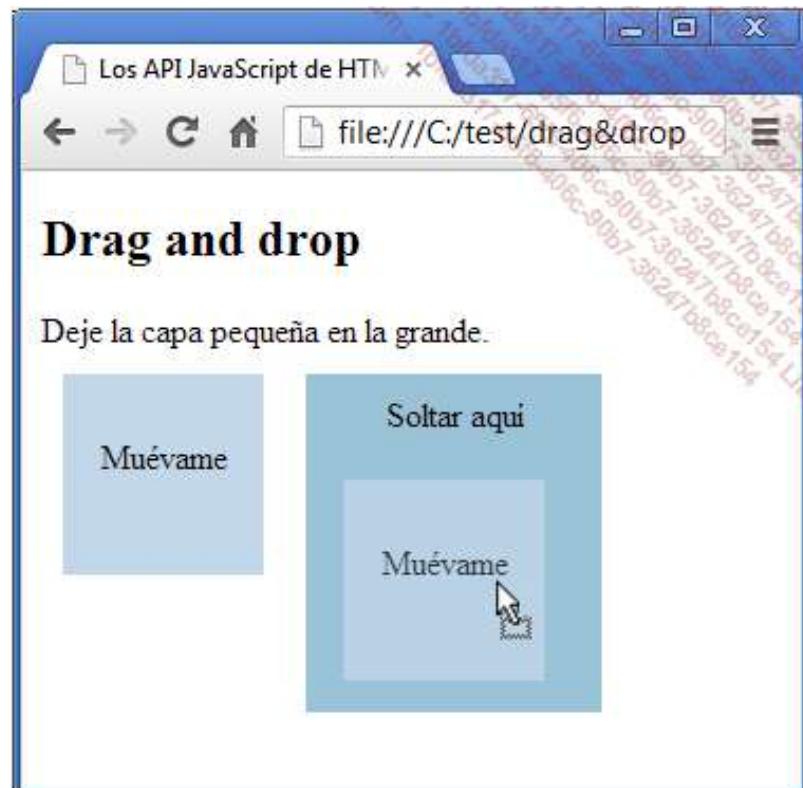
```
function dragOverHandler(event) {  
event.preventDefault();  
return false;  
}
```

Ejemplo 1

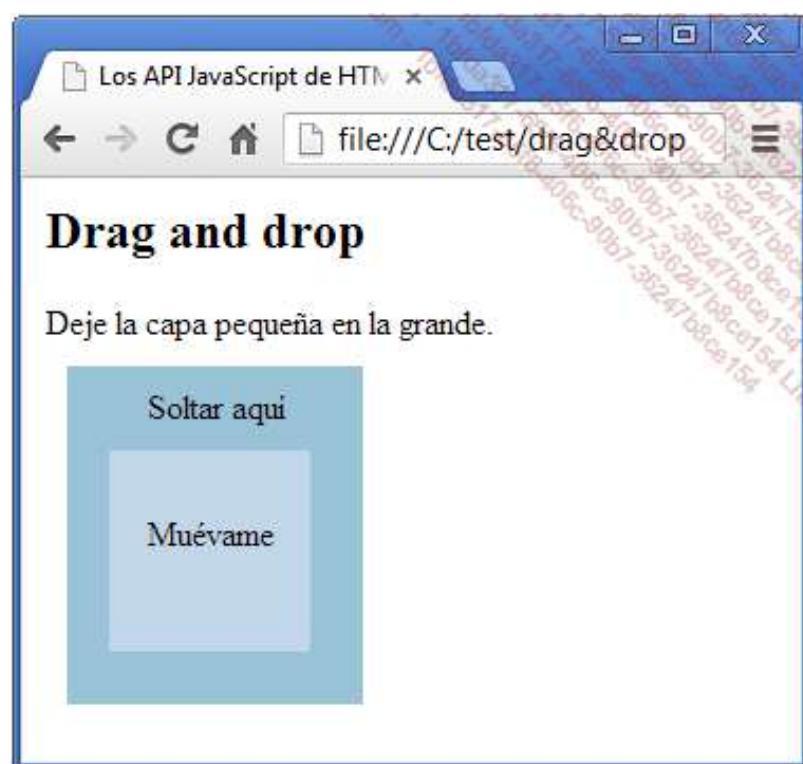
Al abrir el archivo:



Justo antes de soltar el botón del ratón:



Una vez que se ha soltado el elemento:



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#boxA, #boxB { float:left;
padding:10px;
```

```

        margin:10px; }

#boxA { width:75px; height:75px;
         background-color: rgb(195,215,235);
         text-align: center; }

#boxB { width:120px; height:140px;
         background-color: rgb(155,195,215);
         text-align: center; }

</style>
<script type="text/javascript">
function dragStart(ev) {
ev.dataTransfer.effectAllowed='move';
ev.dataTransfer.setData("Text", ev.target.getAttribute('id'));
}
function dragOver(ev) {
event.preventDefault();
return false;
}
function dragDrop(ev) {
var src = ev.dataTransfer.getData("Text");
ev.target.appendChild(document.getElementById(src));
}
</script>
</head>
<body>
<h2>Drag and drop</h2>
<div>Deje la capa pequeña en la grande.</div>
<div id="boxA" draggable="true" ondragstart="dragStart(event)">
<br>Muévame
</div>
<div id="boxB" ondrop="dragDrop(event)"
ondragover="dragOver(event)">
Soltar aquí
</div>
</body>
</html>

```

Comentario

La función `dragStart()` se ha visto en la sección Un primer ejemplo de desplazamiento, de este capítulo.

```
<div id="boxB" ondrop="dragDrop(event)"
ondragover="dragOver(event)">
```

Define la zona donde la capa pequeña se soltará.

```
function dragOver(ev) {
event.preventDefault();
return false;
}
```

Es imprescindible anular (`preventDefault()`) la función `dragOver()` si queremos dejar un elemento.

```
function dragDrop(ev) {
var src = ev.dataTransfer.getData("Text");
ev.target.appendChild(document.getElementById(src));
}
```

La función `dragDrop()` se apropiá (`getData`) del objeto que se mueve y lo añade (`appendChild`) a la capa grande.

Ejemplo 2

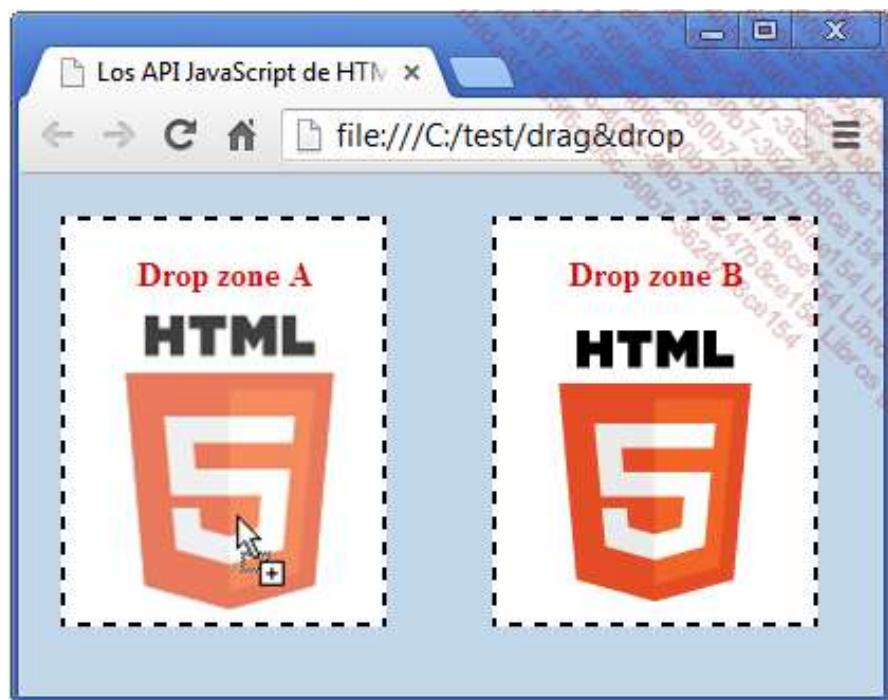
Vamos a hacer un drag&drop con una imagen.



Dejamos la imagen en la zona B.



También es posible dejarla una segunda vez, ahora en la zona A.



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
```

```

<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
body { background: rgb(195,215,235) }
#dropzone { background: white;
            float: left;
            margin-left: 10px;
            text-align: center;
            width: 150px;
            height: 190px;
            border: 2px dashed black;
            color: #d90b0b;
            font-weight:bold; }
#dropzone2 { background: white;
            float: left;
            margin-left: 50px;
            text-align: center;
            width: 150px;
            height: 190px;
            border: 2px dashed black;
            color: #d90b0b;
            font-weight: bold; }
#box { padding-left: 125px; }
</style>
</head>
<body>
<script type="text/javascript">
function dragStart(ev) {
ev.dataTransfer.effectAllowed='copy';
ev.dataTransfer.setData("Text", ev.target.getAttribute('id'));
return true;}
function dragEnter(ev) {
event.preventDefault();
return true;}
function dragOver(ev) {
return false;}
function dragDrop(ev) {
var eleid = ev.dataTransfer.getData("Text");
ev.target.appendChild(document.getElementById(eleid));
ev.preventDefault();
}
</script>
<div id="box">

</div>
<br>
<div id="dropzone" dropzone="copy" ondragenter="return dragEnter(event)"
ondragover=" return dragOver(event)" ondrop="return dragDrop(event)">
<p>Drop zone A</p>
</div>
<div id="dropzone2" ondragenter="return dragEnter(event)"
ondragover=" return dragOver(event)" ondrop="return dragDrop(event)">
<p>Drop zone B </p>
</div>
</body>
</html>

```

Comentario

```

function dragStart(ev) {
ev.dataTransfer.effectAllowed='copy';
ev.dataTransfer.setData("Text", ev.target.getAttribute('id'));
return true;}

```

La función `dragStart()` gestiona el elemento que se va a mover. Determina el efecto asignado al desplazamiento (`effectAllowed='copy'`), pero sobre todo se apropia del elemento desplazable (`setData`).

```
function dragEnter(ev) {  
    event.preventDefault();  
    return true;}
```

Impide el comportamiento por defecto (`preventDefault()`) de `dragEnter()`, pero no lo anula (`return true`).

```
function dragOver(ev) {  
    return false;}
```

Anula (`return false`) la función `dragOver()`.

```
function dragDrop(ev) {  
    var eleid = ev.dataTransfer.getData("Text");  
    ev.target.appendChild(document.getElementById(eleid));  
    ev.preventDefault();  
}
```

La función `dragOver()` se apropia del elemento que se mueve (`getData`) y lo añade (`appendChild`) a la capa.

Ejemplo 3

Ilustremos alguno de los eventos del API Drag and Drop.

Al inicio del desplazamiento:



Cuando el elemento que se mueve entra en la zona de drop:



Cuando termina el drag&drop:



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#console { width: 250px;
            border: 1px solid black;
            padding-left: 5px;
```

```

        margin-bottom: 20px; }

#boxA, #boxB { float: left;
               padding: 25px;
               margin: 5px; }

#boxA { width:20px;
         height:20px;
         background-color: rgb(195,215,235); }

#boxB { width:80px;
         height:80px;
         border: 1px solid black;
         text-align: center;
         margin-left: 45px; }

</style>
<script type="text/javascript">
function dragStart(ev) {
ev.dataTransfer.effectAllowed='move';
ev.dataTransfer.setData("Text", ev.target.getAttribute('id'));
document.getElementById('console').innerHTML = "El movimiento ha comenzado ";
}
function dragEnter(ev) {
document.getElementById('console').innerHTML = "El elemento entra en la zona de drop";
}
function dragLeave(ev) {
document.getElementById('console').innerHTML = "El elemento sale de la zona de drop";
}
function dragOver(ev) {
event.preventDefault();
return false;
}
function dragDrop(ev) {
var src = ev.dataTransfer.getData("Text");
ev.target.appendChild(document.getElementById(src));
}
function dragEnd(ev) {
document.getElementById('console').innerHTML = "El elemento se ha soltado";
}
</script>
</head>
<body>
<h2>Drag and drop</h2>
<div id="console">&nbsp;</div>
<div id="boxA" draggable="true" ondragstart="dragStart(event)">
</div>
<div id="boxB" ondrop="dragDrop(event)"
ondragover="dragOver(event)" ondragend="dragEnd(event)"
ondragenter="dragEnter(event)" ondragleave="dragLeave(event)">
</div>
</body>
</html>
```

Comentario

```

function dragStart(ev) {
ev.dataTransfer.effectAllowed='move';
ev.dataTransfer.setData("Text", ev.target.getAttribute('id'));
document.getElementById('console').innerHTML = "El movimiento ha comenzado ";
}
```

La función dragStart() muestra un primer mensaje cuando un elemento de la lista se empieza a mover.

```
function dragEnter(ev) {  
document.getElementById('console').innerHTML = "El elemento entra  
en la zona de drop";  
}  
function dragLeave(ev) {  
document.getElementById('console').innerHTML = "El elemento sale  
de la zona de drop";  
}
```

Las funcionalidades `dragEnter()` y `dragLeave()` muestran un mensaje cuando la capa entra o sale de la zona de drop.

```
function dragOver(ev) {  
event.preventDefault();  
return false;  
}
```

La anulación del comportamiento por defecto del `dragOver`.

```
function dragDrop(ev) {  
var src = ev.dataTransfer.getData("Text");  
ev.target.appendChild(document.getElementById(src));  
}
```

La función `dragDrop()` gestiona el proceso de soltar el elemento. La variable `src` se apropiá de (`getData`) la capa que se mueve. Después el script añade (`appendChild`) el elemento a la capa grande.

```
function dragEnd(ev) {  
document.getElementById('console').innerHTML = "El elemento se ha  
soltado";  
}
```

La función `dragEnd()` muestra un último mensaje cuando termina el desplazamiento.

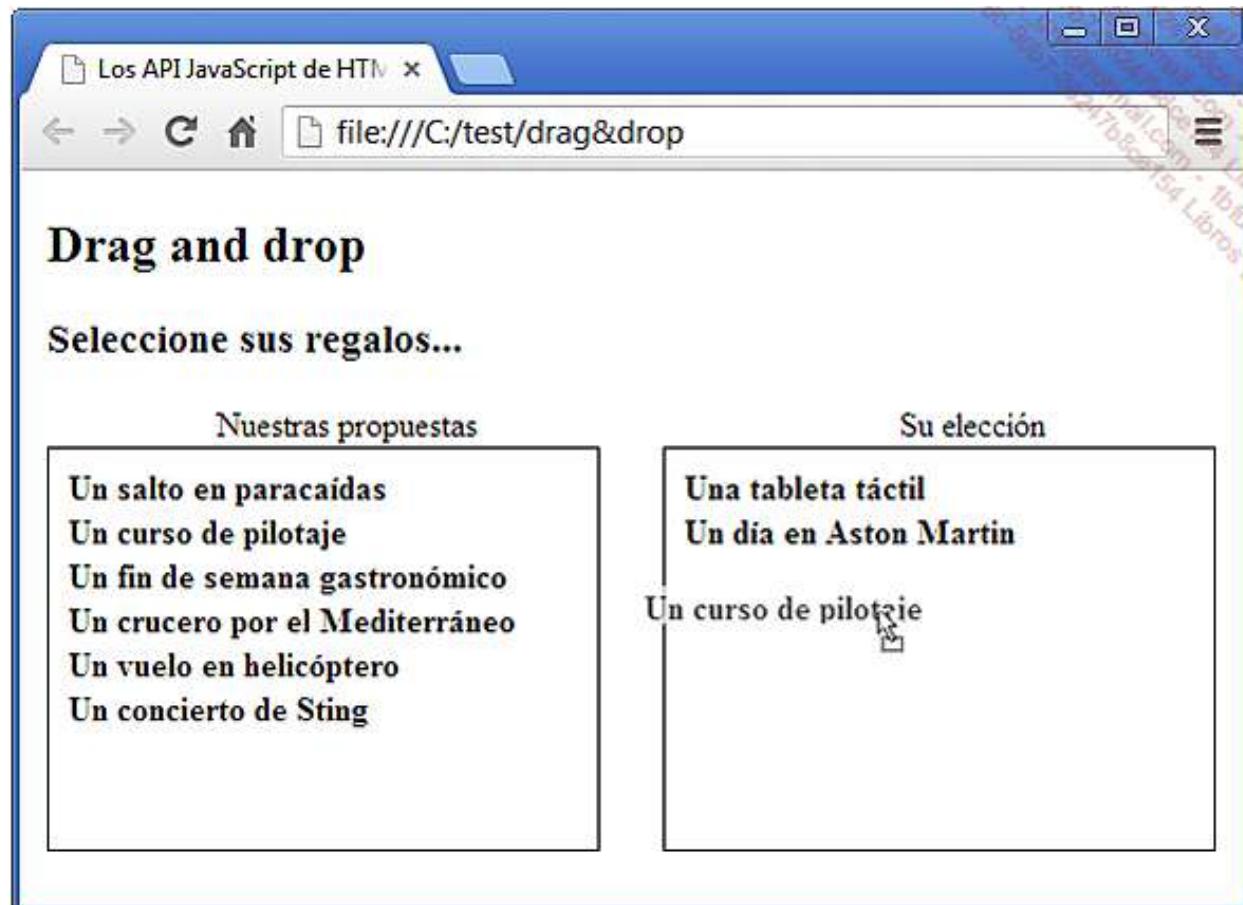
Una aplicación final

Presentamos un drag&drop a partir de una lista de sugerencias de regalos.

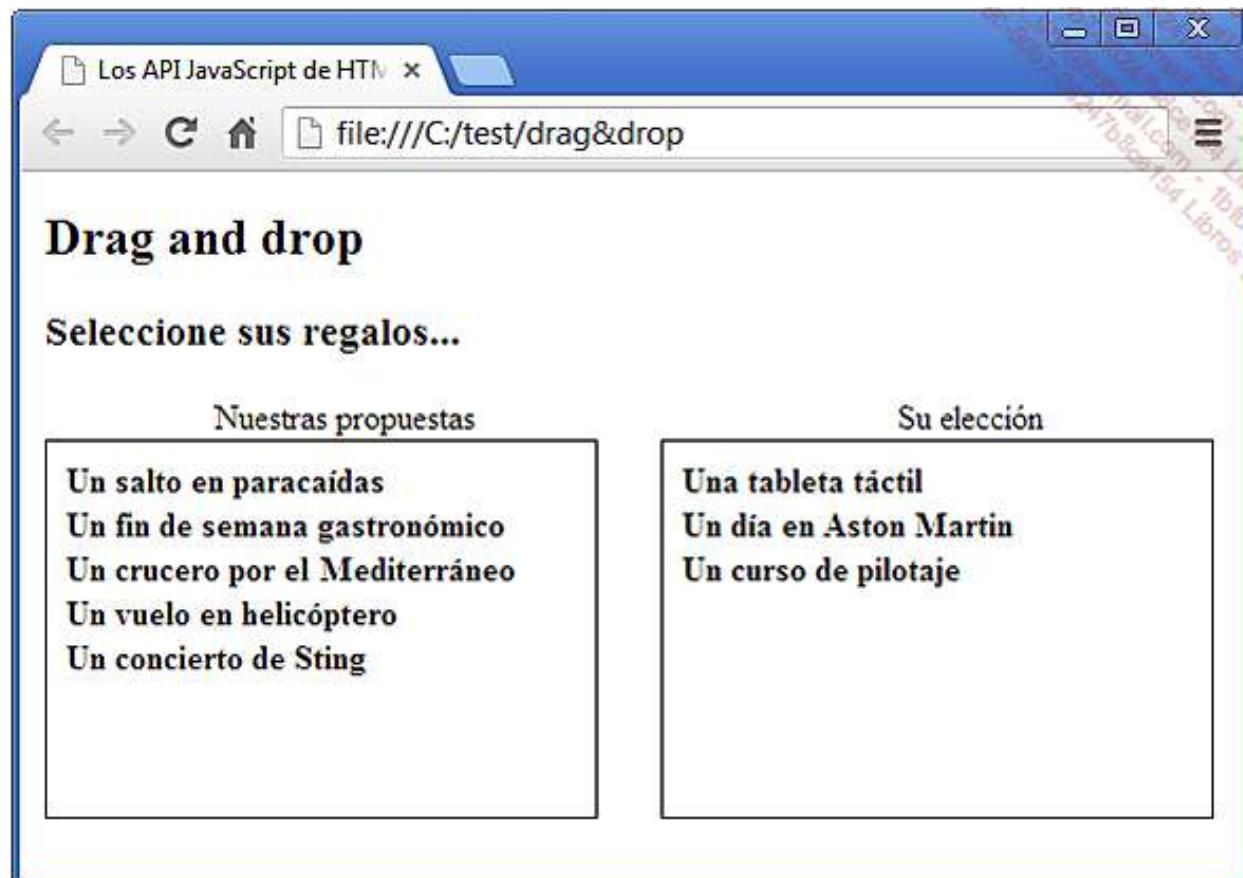
Al cargar la página:



Para seleccionar una de las opciones que se proponen, es suficiente con mover la opción a la capa prevista para ello.



También es posible revisar el orden de prioridad de las elecciones y, si es necesario, para los indecisos, volver a dejar una propuesta en su capa inicial.



```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#boxA { float: left;
         width: 250px;
         height: 180px;
         padding: 5px;
         margin-right: 30px;
         border: 1px solid black;
         background-color: white; }
#boxB { float: left;
         width: 250px;
         height: 180px;
         padding: 5px;
         border: 1px solid black;
         background-color: white; }
.word { width: 220px;
        height: 16px;
        padding: 1px;
        margin: 3px;
        font-weight: bold;
        background-color: white;
        display: block; }
#span1 { margin-left: 80px; }
#span2 { margin-left: 200px; }
*[draggable=true] {
-moz-user-select:none;
-khtml-user-drag: element;
cursor: move; }
</style>
<script type="text/javascript">
function dragStart(ev) {
ev.dataTransfer.effectAllowed = 'move';
ev.dataTransfer.setData("text/plain",
ev.target.getAttribute('id'));
return true;
}
function dragOver(ev) {
ev.preventDefault();
}
function dragEnd(ev) {
return true;
}
function dragDrop(ev) {
var idDrag = ev.dataTransfer.getData("Text");
ev.target.appendChild(document.getElementById(idDrag));
ev.preventDefault();
}
</script>
</head>
<body>
<header>
<h2>Drag and drop</h2>
</header>
<section>
<h3>Seleccione sus regalos...</h3>
<div><span id="span1">Nuestras propuestas </span><span id="span2">
Su elección</span></div>
<div id="boxA" ondragover="dragOver(event)"
ondrop="dragDrop(event)">
<div id="word1" class="word" draggable="true" ondragstart="return
dragStart(event)" ondragend="dragEnd(event)">Una tableta
```

```

táctil</div>
<div id="word2" class="word" draggable="true" ondragstart="return
dragStart(event)" ondragend="dragEnd(event)">Un salto en paracaídas </div>
<div id="word3" class="word" draggable="true" ondragstart="return
dragStart(event)" ondragend="dragEnd(event)">Un día en Aston Martin</div>
<div id="word4" class="word" draggable="true" ondragstart="return
dragStart(event)" ondragend="dragEnd(event)">Un curso de pilotaje</div>
<div id="word5" class="word" draggable="true" ondragstart="return
dragStart(event)" ondragend="dragEnd(event)">Un fin de semana
gastronómico</div>
<div id="word6" class="word" draggable="true" ondragstart="return
dragStart(event)" ondragend="dragEnd(event)">Un crucero por el
Mediterráneo</div>
<div id="word7" class="word" draggable="true" ondragstart="return
dragStart(event)" ondragend="dragEnd(event)">Un vuelo en
helicóptero</div>
<div id="word8" class="word" draggable="true" ondragstart="return
dragStart(event)" ondragend="dragEnd(event)">Un concierto de Sting
</div>
</div>
<div id="boxB" ondragover="return dragOver(event)"
ondrop="dragDrop(event)"></div>
</section>
</body>
</html>

```

Comentario

```

function dragStart(ev) {
ev.dataTransfer.effectAllowed = 'move';
ev.dataTransfer.setData("text/plain",
ev.target.getAttribute('id'));
return true;
}

```

La función dragStart () se llama cuando un elemento de la lista empieza a moverse.

```

function dragOver(ev) {
ev.preventDefault();
}

```

El comportamiento por defecto del dragOver se anula para permitir el drop.

```

function dragEnd(ev) {
return true;
}

```

La función dragEnd () se llama cuando el desplazamiento termina.

```

function dragDrop(ev) {
var idDrag = ev.dataTransfer.getData("Text");
ev.target.appendChild(document.getElementById(idDrag));
ev.preventDefault();
}

```

La función dragDrop () gestiona la función del drop del elemento. La variable idDrag identifica y se apropiá (getData) del elemento que se mueve. El script añade (appendChild) el elemento a la capa sobre la que se encuentra.

Presentación y objetivos

El objetivo del API File es modernizar y extender las funcionalidades del antiguo campo de formulario <input type="file"> que existe desde el inicio del lenguaje Html y cuya eficacia no siempre ha estado asegurada.



Este API permite seleccionar archivos desde el sistema del usuario (como con la etiqueta <input type="file">), pero de manera más moderna, por un drag&drop aplicado al archivo seleccionado. Dicho de otra manera, proporciona información útil del archivo seleccionado, como el nombre, tamaño y formato. El objetivo final de este API, además de la gestión, será la administración del *upload* de estos archivos.

En este capítulo solo trataremos la selección de los archivos y la información que administra.

El API File tiene diferentes módulos:

- **File** (que ha dado su nombre al API). Este módulo gestiona el proceso de selección de los archivos y proporciona información en modo de solo lectura, como el nombre del archivo, su tamaño y su tipo MIME.
- **FileList**, una matriz de tipo Array que incluye los objetos correspondientes a los archivos seleccionados
- **Blob**, que permite dividir un archivo en franjas de bits.

Cuando termina la selección, el API File pasa el control al módulo FileReader para leer, de manera asíncrona, el contenido del archivo en memoria. Esto inicia un objeto FileReader cuya propiedad **result** se puede usar para acceder a los datos del archivo.

FileReader tiene cuatro opciones para leer un archivo:

- `FileReader.readAsBinaryString(Blob|File)`. La propiedad **result** contendrá los datos en forma de cadena de caracteres binarios.
- `FileReader.readAsText(Blob|File, opt_encoding)`. La propiedad **result** contendrá los datos en forma de cadena de caracteres de texto. Por defecto, esta cadena de caracteres está en UTF-8.
- `FileReader.readAsDataURL(Blob|File)`. La propiedad **result** contendrá los datos en forma de una dirección URL.
- `FileReader.readAsArrayBuffer(Blob|File)`. La propiedad **result** contendrá los datos en forma de un objeto ArrayBuffer.

No hay que confundir el API File con el API Filesystem. Este último permitirá, en su estado final de desarrollo, organizar un sistema de directorios y archivos, como en las aplicaciones de escritorio. De este modo, son posibles la creación, lectura y eliminación de directorios, así como la creación, lectura, duplicación y eliminación de archivos. Permitirá también copiar, renombrar, duplicar los directorios y archivos de este sistema.

Los ejemplos y otros desarrollos necesitan que los archivos estén ubicados en un servidor local o en línea.

Disponibilidad del API

En lo que respecta a los navegadores de escritorio, este API está disponible en:

- Internet Explorer 10+.
- Firefox 3.6+
- Safari 5.1+
- Chrome 6+
- Opera 11.1+

Para los Smartphone y tabletas, estos dos navegadores soportan el API:

- Opera Móvil 11.1
- Android 3.0

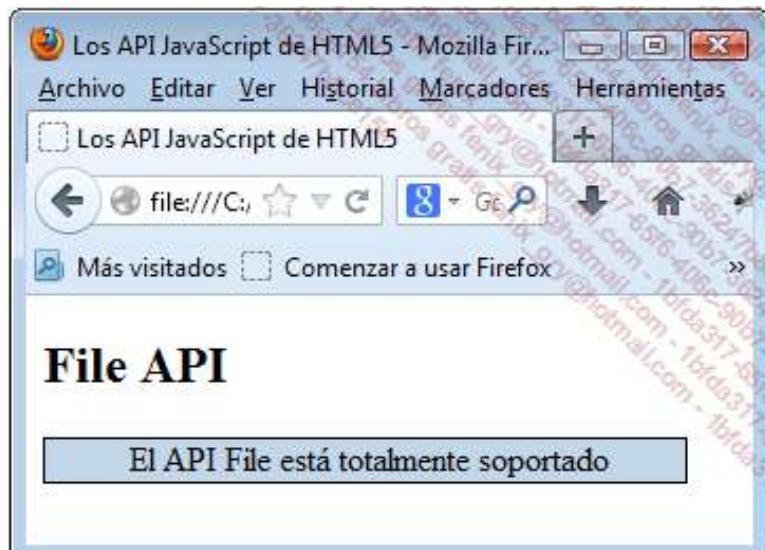
Para probar el soporte de este API, se aconseja probar no solo su presencia (`window.File`), sino también sus diferentes componentes (`window.FileReader`, `window.FileList` y `window.Blob`).

La prueba tiene la siguiente forma:

```
if (window.File && window.FileReader && window.FileList &&
window.Blob) {
// El navegador soporta el File API y sus componentes.
}
```

El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#box { width: 300px;
        border: 1px solid black;
        background-color: rgb(195,215,235);
        padding-left: 3px;
        text-align: center;}
</style>
<script type="text/javascript">
function init(){
if (window.File && window.FileReader && window.FileList &&
window.Blob){
msg = "El API File está totalmente soportado";
document.querySelector('#box').innerHTML = msg;
}
else {
msg = "El API File no está totalmente soportado";
document.querySelector('#box').innerHTML = msg;
}
}
</script>
</head>
<body onload="init();">
<h2>File API</h2>
<div id="box">&nbsp;</div>
</body>
</html>
```



Selección de archivos usando un formulario

Seleccionar un archivo solo con el API File es sencillo.

```
<input type="file" id="input" onchange="handleFiles(this.files)">
```

donde handleFiles(this.files) es una función cualquier que gestiona los archivos seleccionados.

Cuando el usuario selecciona un archivo, la función handleFiles() llama a un objetoFileList que contiene el objeto File que representa al archivo seleccionado.

Si el desarrollador desea dejar al usuario la posibilidad de seleccionar varios archivos, se debe añadir el atributo multiple al campo del formulario.

```
<input type="file" id="input" multiple  
onchange="handleFiles(this.files)">
```

En este caso, la lista de archivos (FileList) que se pasa a la función handleFiles() contiene un objeto File para cada archivo seleccionado.

Ejemplo

Vamos a seleccionar un archivo situado en nuestro sistema usando un campo de formulario de selección de archivos, pero llamando al API File.

Al inicio del ejemplo:



Une vez elegido el archivo:



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
ul { width: 250px; }
#file-list { list-style: none; }
#file-list li { border: 1px solid black;
               padding-left: 5px;
               margin-left: -40px; }
</style>
</head>
<body>
<h2>File API</h2>
<p>Seleccione uno o varios archivos</p>
<p>
<form>
<input id="files-upload" type="file" multiple>
</form>
</p>
<h3>Archivos cargados</h3>
<ul id="file-list">
<li class="no-items">Todavía no hay archivos seleccionados
</li>
</ul>
<script type="text/javascript">
(function () {
var filesUpload = document.getElementById("files-upload");
var fileList = document.getElementById("file-list");
function traverseFiles(files) {
var li;
var file;
```

```

var fileInfo;
fileList.innerHTML = "";
for (var i=0, il=files.length; i<il; i++) {
li = document.createElement("li");
file = files[i];
fileInfo = "<div><b>Nombre:</b> " + file.name + "</div>";
fileInfo += "<div><b>Tamaño:</b> " + file.size + " bytes</div>";
fileInfo += "<div><b>Tipo:</b> " + file.type + "</div>";
li.innerHTML = fileInfo;
fileList.appendChild(li);
};
};

filesUpload.onchange = function () {
traverseFiles(this.files);
};

})();
</script>
</body>
</html>

```

Comentario

```
function traverseFiles(files) {
```

La función `traverseFiles()` es una función equivalente a la de `handleFiles()` de nuestra presentación (ver más atrás).

```

var li;
var file;
var fileInfo;
```

Definición de una serie de variables.

```
for (var i=0, il=files.length; i<il; i++) {
```

Un bucle `for` recorre los archivos `files` correspondientes a los archivos seleccionados por el usuario y los pasa como argumento de la función `traverseFiles()`.

```
li = document.createElement("li");
```

Para cada objeto, se crea un ítem de lista.

```
file = files[i];
```

La variable `file` contiene el archivo `files` en cada pasada del bucle.

```

fileInfo = "<div><b>Nombre:</b> " + file.name + "</div>";
fileInfo += "<div><b>Tamaño:</b> " + file.size + " bytes</div>";
fileInfo += "<div><b>Tipo:</b> " + file.type + "</div>";
```

La variable `fileinfo` contiene diversas propiedades del objeto `file`, como el nombre (`file.name`), el tamaño (`file.size`) y el formato (`file.type`).

```

li.innerHTML = fileInfo;
fileList.appendChild(li);
```

Estas líneas muestran esta información.

```

filesUpload.onchange = function () {
traverseFiles(this.files);
};
```

Cada vez que el usuario selecciona un archivo (`filesUpload.onchange`), se llama a la función `traverseFiles()` detallada más atrás. Observe que el archivo es un argumento de la función (`this.files`).

Selección de archivos por drag&drop

La selección de los archivos también se puede hacer por medio de un drag&drop.

Ejemplo

Además del elemento seleccionado por el campo de formulario, ofrecemos al usuario la posibilidad de realizar esta selección por medio de un drag&drop.

Al inicio:



Se hace un drag&drop.



El resultado final:



El código

```
<!DOCTYPE html>
<html lang="es">
```

```
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#drop-area { width: 250px;
              border: 2px dashed #ddd;
              padding: 10px;
              margin-bottom: 1em; }
ul { width: 250px; }
#file-list { list-style: none; }
#file-list li { border: 1px solid black;
                  padding-left: 5px;
                  margin-left: -40px; }
</style>
</head>
<body>
<h2>File API</h2>
<p>Seleccione uno o varios archivos para drag&drop</p>
<p>
<form>
<input id="files-upload" type="file" multiple>
</form>
</p>
<p id="drop-area">
Haga Drag&Drop de los archivos aquí.
</p>
<h3>Archivos cargados </h3>
<ul id="file-list">
<li class="no-items">Todavía no hay archivos seleccionados
</li>
</ul>
<script type="text/javascript">
(function () {
var filesUpload = document.getElementById("files-upload");
var dropArea = document.getElementById("drop-area");
var fileList = document.getElementById("file-list");
function traverseFiles (files) {
var li;
var file;
var fileInfo;
fileList.innerHTML = "";
for (var i=0, il=files.length; i<il; i++) {
li = document.createElement("li");
file = files[i];
fileInfo = "<div><b>Nombre:</b> " + file.name + "</div>";
fileInfo += "<div><b>Tamaño:</b> " + file.size + " bytes</div>";
fileInfo += "<div><b>Tipo:</b> " + file.type + "</div>";
li.innerHTML = fileInfo;
fileList.appendChild(li);
};
};
filesUpload.onchange = function () {
traverseFiles(this.files);
};
dropArea.ondragenter = function () {
return false;
};
dropArea.ondragover = function () {
return false;
};
dropArea.ondrop = function (evt) {
traverseFiles(evt.dataTransfer.files);
return false;
};
})();
</script>
```

```
</body>
</html>
```

Comentario

La primera parte del script es igual que el ejemplo anterior. Nos centraremos en la parte relativa al drag&drop.

```
dropArea.ondragenter = function () {
return false;
};
dropArea.ondragover = function () {
return false;
};
```

Anula el evento dragenter y el dragover para permitir dejar el archivo.

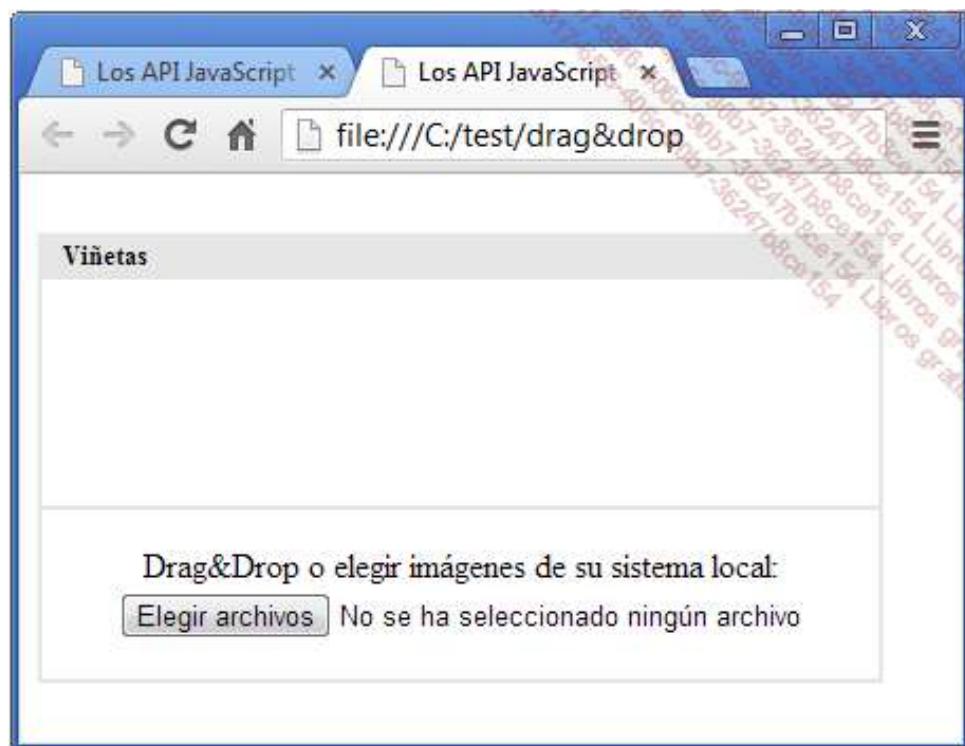
```
dropArea.ondrop = function (evt) {
traverseFiles(evt.dataTransfer.files);
return false;
};
```

Cuando se ha soltado el archivo (ondrop), se llama a la función traverseFiles() a la que se le pasa como argumento el archivo en cuestión (evt.dataTransfer.files).

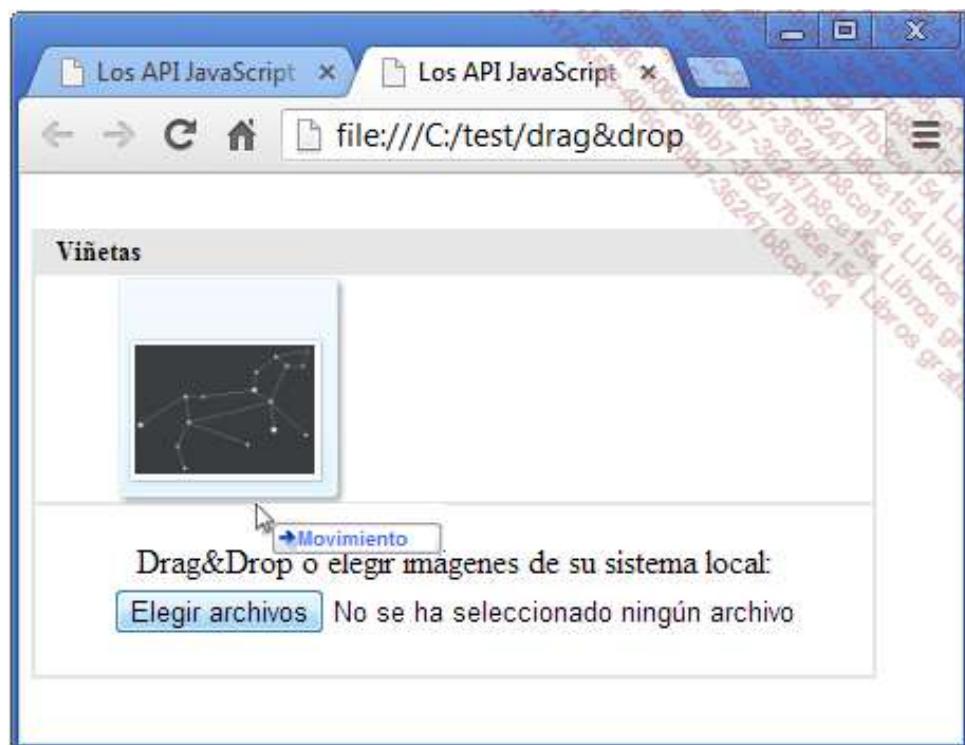
Aplicación final

Presentamos otro ejemplo, este más concretamente dedicado a las imágenes, porque permite visualizar una viñeta de la imagen seleccionada.

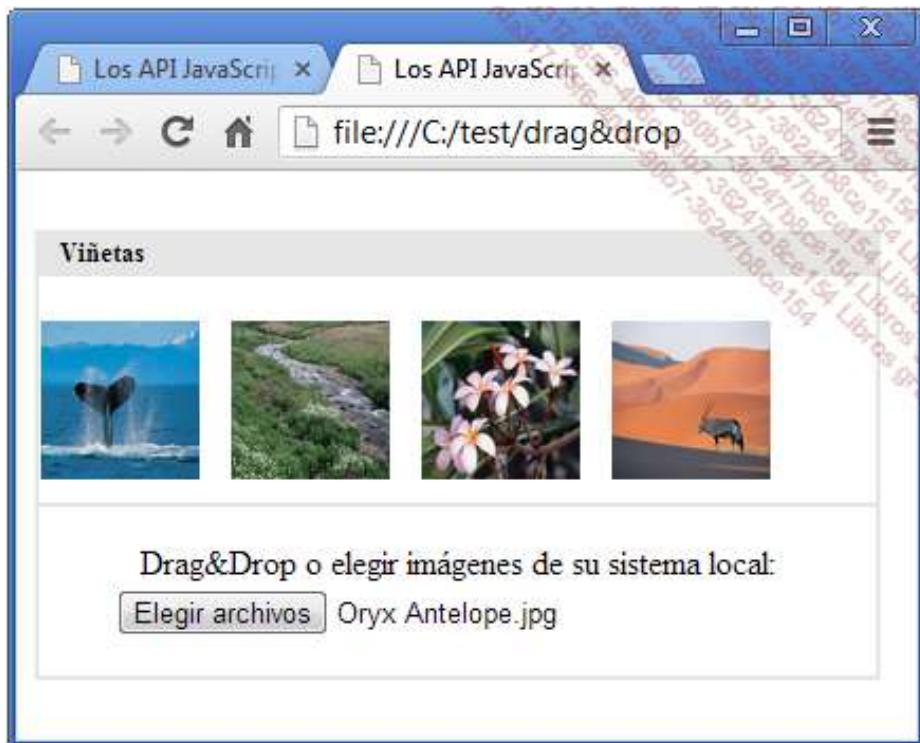
Al cargar la página:



Una imagen se añade usando un drag&drop.



Al final, después de cargar cuatro imágenes:



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
.tableheader { background-color: rgb(230,230,230);
    padding-left: 10px;
    font-weight: bold;
    font-size: 12px;}
.tablecontent { text-align: center;}
.images { height: 75px;
    width: 75px;
    border: 0px;
    margin: 15px 15px 0 0;}
table, td { border-color: rgb(230,230,230);
    border-style: solid;}
</style>
<script type="text/javascript">
function ImagesSelected(myFiles) {
for (var i = 0, f; f = myFiles[i]; i++) {
var imageReader = new FileReader();
imageReader.onload = (function(aFile) {
return function(e) {
var span = document.createElement('span');
span.innerHTML = [''].join('');
document.getElementById('thumbs').insertBefore(span, null);
};
})(f);
imageReader.readAsDataURL(f);
}
}
function dropIt(e) {
imagesSelected(e.dataTransfer.files);
e.stopPropagation();
e.preventDefault();
```

```

}
</script>
</head>
<body>
<br>
<table width="400" border="1" cellspacing="0">
<tr>
<td class="tableheader">
Viñetas
</td>
</tr>
<tr>
<td height="105" ondragenter="return false" ondragover="return
false" ondrop="dropIt(event)">
<output id="thumbs"></output>
</td>
</tr>
<tr>
<td class="tablecontent">
<p>
Drag&drop o elegir imágenes de su sistema local:
<input type="file" id="input" size="10" multiple="true"
onchange="imagesSelected(this.files)">
</p>
</td>
</tr>
</table>
</body>
</html>

```

Comentario

```
function ImagesSelected(myFiles) {
```

La función ImagesSelected() se llama cuando el usuario ha seleccionado las imágenes. Observe que estas imágenes son los argumentos de la función (myFiles).

```
for (var i = 0, f; f = myFiles[i]; i++) {
```

Un bucle for recorre estas imágenes. Observe que la variable f, en forma de una variable Array, contiene las diferentes imágenes.

```
var imagenReader = new FileReader();
```

La variable imagenReader llama al módulo FileReader.

```

return function(e) {
var span = document.createElement('span');
span.innerHTML = [''].join('');
document.getElementById('thumbs').insertBefore(span, null);
};
})(f);
imageReader.readAsDataURL(f);
}
}
```

Esta función realiza la visualización de las viñetas. Apuntamos a la dirección de las imágenes. FileReader ha extraído la URL con imagenReader. readAsDataURL(f). Esta se transmite a la etiqueta imagen por la propiedad result (e.target.result).

```
function dropIt(e) {
imagesSelected(e.dataTransfer.files);
```

```
e.stopPropagation();
e.preventDefault();
}
```

Como habrá adivinado, esta función dropIt() se encarga de terminar el drag&drop. Llama a la función ImagesSelected() proporcionándole los archivos de imagen como argumento (e.dataTransfer.files).

Presentación y objetivos

El API Html5 Web Messaging permite transmitir cadenas de caracteres (*string*) entre diferentes ventanas o etiquetas `<iframe>`. Estas pueden estar localizadas en el mismo dominio o en dominios diferentes.

El JavaScript clásico no puede acceder al código situado en un dominio (o subdominio) diferente, un protocolo distinto (http no es idéntico a https) o un puerto. Por tanto, no es posible ninguna comunicación entre dominios diferentes.

Esto ahora está permitido gracias a este API Web Messaging.

Las pruebas y otros desarrollos se deben hacer en línea o pasando por un servidor local. Hemos optado por poner los ejemplos en línea, ya que será necesario simular dominios diferentes en el servidor local, lo que implica una cantidad de operaciones importante, desproporcionada en relación con los ejemplos de este capítulo.

Disponibilidad del API

Este API está soportado por los navegadores de escritorio:

- Internet Explorer 8.0+
- Firefox 3.0+
- Safari 4.0+
- Google Chrome 2.0+
- Opera 9.6+

Para los Smartphone y tabletas:

- iOS Safari 3.2
- Opera Móvil 10.0
- Android 2.1

A nivel de código, comprobamos si el método `postMessage` existe o no en el objeto `window`, usando el operador `typeof`.

```
typeof window.postMessage
```

Ejemplo

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#box { width: 270px;
        border: 1px solid black;
        background-color: rgb(195,215,235) ;
        padding-left: 3px;
        text-align: center; }
</style>
<script type="text/javascript">
function init(){
if(typeof window.postMessage === "undefined") {
msg = "El Web Messaging no está soportado";
document.querySelector('#box').innerHTML = msg;
}
else {
msg = "El Web Messaging está soportado";
document.querySelector('#box').innerHTML = msg;
}
}
</script>
</head>
<body onload="init();">
<h2>Web Messaging</h2>
<div id="box">&nbsp;</div>
</body>
</html>
```



Enviar un mensaje (postMessage)

La comunicación pasa por el método `postMessage()`, cuya sintaxis es:

```
postMessage(dato, destino)
```

donde `dato` contiene el mensaje enviado y `destino` indica la dirección absoluta del sitio que recibe el mensaje.

Las URL siempre se deben expresar como direcciones absolutas.

No hay que confundirse con el método `postMessage()` de Workers, que permite la comunicación en segundo plano entre la página principal y el worker JavaScript (cf. capítulo El JavaScript en segundo plano - Comunicación con el script Workers (`postMessage`)).

1. En el mismo dominio

Comenzamos con ejemplos donde una página y un marco en línea (`<iframe>`) se sitúan en el mismo dominio.

Ejemplo 1

Sea una página que contiene un marco en línea. Enviamos un mensaje de la página a este iframe.

La página en su estado inicial:



La misma página después de enviar el mensaje:



El código de la página principal (ejemplo1.htm)

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
iframe { width: 350px;
          height: 100px;
          margin: 10px auto;
          border: 1px solid #000; }
</style>
<script type="text/javascript">
window.onload = function () {
var iframeWin = document.getElementById("marco").contentWindow;
var form = document.getElementById("formulario");
var miMensaje = document.getElementById("mi-mensaje");
miMensaje.select();
form.onsubmit = function () {
iframeWin.postMessage(miMensaje.value, "http://www.lehtml.com");
return false;
}
}
</script>
</head>
<body>
<header>
<h2>Web Messaging</h2>
</header>
<section>
<form id="formulario" action="">
<input type="text" id="mi-mensaje" value="Su mensaje aquí..."/>
<input type="submit" value="postMessage"/>
</form>
<iframe id="marco" src="postMessaging.htm"></iframe>
</body>
</html>
```

Comentario

```
form.onsubmit = function () {
```

```

iframeWin.postMessage(miMensaje.value, "http://www.lehtml.com");
return false;
};

```

El script envía un mensaje (`postMessage()`) a la página contenida en la etiqueta `<iframe>` (variable `iframeWin`). Esta función `postMessage()` tiene dos argumentos: el contenido de la zona de texto (`miMensaje.value`) y el dominio destino (en nuestro ejemplo, `http://www.lehtml.com`).

El código del iframe (postMessaging.htm)

```

<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<script type="text/javascript">
function verMensaje (evt) {
var message;
if (evt.origin !== "http://www.lehtml.com") {
message = "No es bienvenido aquí";
}
else {
message = "Recibido " + evt.data + " de " + evt.origin;
}
document.getElementById("mensaje-recibido").innerHTML = message;
}
if (window.addEventListener) {
window.addEventListener("message", verMensaje, false);
}
else {
window.attachEvent("onmessage", verMensaje);
}
</script>
</head>
<body>
<p id="mensaje-recibido">Todavía no he recibido el mensaje.</p>
</body>
</html>

```

Comentario

```

if (evt.origin !== "http://www.lehtml.com") {
message = " No es bienvenido aquí ";
}

```

El script hace un control en el dominio de origen para verificar si proviene del dominio que se indica en el código de la página principal (`evt.origin !== "..."`). Se aconseja a los desarrolladores verificar el atributo `origin` para asegurarse de que los mensajes se envían desde el dominio esperado.

```

else {
message = "Recibido " + evt.data + " de " + evt.origin;
}

```

El mensaje se muestra (`evt.data`), así como su origen (`evt.origin`).

```

if (window.addEventListener) {
window.addEventListener("message", verMensaje, false);
}
else {
window.attachEvent("onmessage", verMensaje);
}

```

Dos fórmulas del manejador de eventos para que el script sea compatible.

Ejemplo 2

Hagamos el camino inverso. Enviamos un mensaje del iframe a la página principal.

Cuando se abre la página principal:



Cuando se recibe el mensaje enviado desde el marco en línea:



El código de la página principal (ejemplo2.htm)

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
iframe { width: 350px;
          height: 100px;
          margin: 10px auto;
          border: 1px solid #000; }

</style>
<script type="text/javascript">
function verMensaje (evt) {
var message;
if (evt.origin !== "http://www.lehtml.com") {
```

```

message = "No es bienvenido aquí";
}
else {
message = "Recibido desde el iframe " + evt.data + " de " + evt.origin;
}
document.getElementById("mensaje-recibido").innerHTML = message;
}
if (window.addEventListener) {
window.addEventListener("message", verMensaje, false);
}
else {
window.attachEvent("onmessage", verMensaje);
}
</script>
</head>
<body>
<p id="mensaje-recibido">Todavía no he recibido el mensaje.</p>
<iframe id="marco" src="postMessaging2.htm"></iframe>
</body>
</html>

```

Comentario

```

if (evt.origin !== "http://www.lehtml.com") {
message = "No es bienvenido aquí";
}

```

Control en el dominio de origen del mensaje.

```

Else {
message = "Recibido desde el iframe " + evt.data + " de " + evt.origin;
}

```

En caso afirmativo, visualización del mensaje enviado por el iframe.

El código del iframe (postMessaging2.htm)

```

<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<script type="text/javascript">
window.onload = function () {
var form = document.getElementById("formulario");
var miMensaje = document.getElementById("mi-mensaje");
miMensaje.select();
form.onsubmit = function () {
parent.postMessage(miMensaje.value, "http://www.lehtml.com");
return false;
}
}
</script>
</head>
<body>
<form id="formulario" action="">
<input type="text" id="mi-mensaje" value="Su mensaje aquí..."><br>
<input type="submit" value="postMessage">
</form>
</body>
</html>

```

Comentario

```

form.onsubmit = function () {
parent.postMessage(miMensaje.value, "http://www.lehtml.com");
return false;
}

```

El script envía un mensaje (`postMessage()`) a la página principal que es padre del iframe que la contiene (`parent`). Esta función `postMessage()` tiene dos argumentos: el contenido de la zona de texto (`miMensaje.value`) y el dominio de destino (en nuestro ejemplo, `http://www.lehtml.com`).

2. En otro dominio

La verdadera novedad del API Web Messaging es que esta comunicación se puede realizar entre la página principal y el iframe alojados en dominios totalmente diferentes. En nuestro ejemplo, la página principal se sitúa en el dominio `www.doscollection.be` y la página del iframe en el dominio `www.lehtml.com`.

No hay modificaciones importantes en el código. Sin embargo, es necesario tener cuidado con la adaptación del dominio de destino del mensaje.



El código de la página principal (ejemplo3.htm)

```

<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
iframe { width: 350px;
          height: 100px;
          margin: 10px auto;
          border: 1px solid #000; }
</style>
<script type="text/javascript">
window.onload = function () {
var iframeWin = document.getElementById("marco").contentWindow;
var form = document.getElementById("formulario");
var miMensaje = document.getElementById("mi-mensaje");
miMensaje.select();
form.onsubmit = function () {

```

```

        iframeWin.postMessage(miMensaje.value,
        "http://www.lehtml.com");
    return false;
};

};

</script>
</head>
<body>
<header>
<h2>Web Messaging</h2>
</header>
<section>
<form id="formulario" action="">
<input type="text" id="mi-mensaje" value="Su mensaje aquí...">
<input type="submit" value="postMessage">
</form>
<iframe id="marco"
src="http://localhost:8100/onemore/ejercicio1.html"/eni/
postMessaging3.htm"></iframe>
</body>
</html>

```

Comentario

```

<iframe id="marco"
src="http://www.lehtml.com"/eni/
postMessaging3.htm"></iframe>

```

En el código Html, el iframe se sitúa correctamente en el dominio www.lehtml.com.

```

form.onsubmit = function () {
  iframeWin.postMessage(miMensaje.value, "http://www.lehtml.com");
  return false;
};

```

En el script, el destino del método postMessage () es www.lehtml.com.

El código del iframe (postMessaging3.htm)

```

<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<script type="text/javascript">
function verMensaje (evt) {
var message;
if (evt.origin !== "http://www.doscollection.be") {
message = "No es bienvenido aquí";
}
else {
message = "Recibido " + evt.data + " de " + evt.origin;
}
document.getElementById("mensaje-recibido").innerHTML = message;
}
if (window.addEventListener) {
window.addEventListener("message", verMensaje, false);
}
else {
window.attachEvent("onmessage", verMensaje);
}
</script>
</head>
<body>

```

```
<p id="mensaje-recibido">Todavía no he recibido el mensaje.</p>
</body>
</html>
```

Comentario

```
if (evt.origin !== "http://www.doscollection.be") {
  message = "No es bienvenido aquí";
}
```

Como la comunicación se establece entre dominios diferentes, es más útil que nunca, incluso indispensable, por razones evidentes de seguridad, verificar el origen del dominio.

```
else {
  message = "Recibido " + evt.data + " de " + evt.origin;
}
```

Si el control es positivo, el mensaje (evt.data) y el origen (evt.origin) se muestran.

Consideraciones de seguridad

Estas entradas externas inquietan a algunos desarrolladores. El W3C ha tomado medidas de seguridad que podemos considerar satisfactorias.

En primer lugar, el concepto origen (*origin*) incluye tres condiciones: el nombre del dominio, el protocolo y el puerto. Así, una página `https://www.abcd.com` tiene un origen diferente que la página `http://www.abcd.com`. Por el contrario, la ruta de acceso no se tiene en cuenta en el concepto de origen (simplemente el dominio). Por tanto, una página a `http://www.abcd.com/index.htm` y otra a `http://www.abcd.com/page.htm` tienen el mismo origen, ya que el dominio es idéntico y solo las rutas son diferentes.

También hemos señalado en los ejemplos que es posible, usando código JavaScript, verificar que el mensaje recibido proviene del origen deseado por el desarrollador. Parece (porque la documentación todavía está inacabada) que, si el origen del mensaje no corresponde al especificado en `postMessage`, el navegador no lo envía.

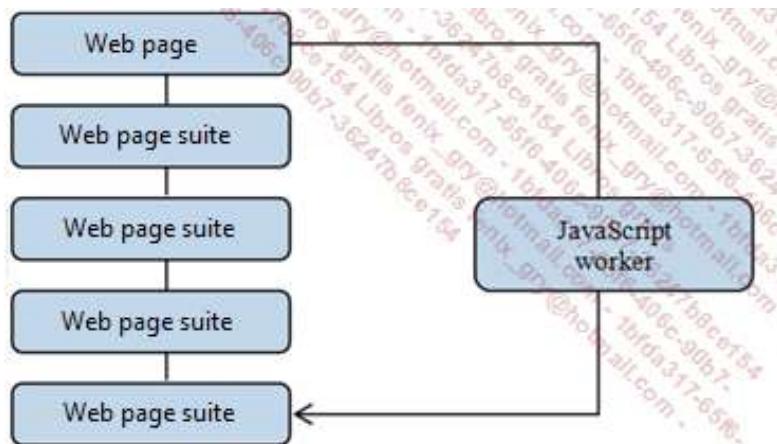
Las personas suspicaces o simplemente prudentes pueden seguir estos consejos:

- Recordemos que es importante controlar si el origen del mensaje corresponde al definido en el código (`evt.origin != "http://nombre_delDominio"`).
- Preferir `element.textContent = e.data` para visualizar los mensajes externos a `element.innerHTML = e.data`, que evalúa `e.data` como etiqueta.
- Evitar el uso de la función `eval`.
- Por motivos de seguridad, utilizar JSON para los mensajes enviados.

Presentación y objetivos

Los Web Workers ofrecen la posibilidad de cargar dinámicamente un archivo JavaScript y ejecutarlo en segundo plano, es decir, en un flujo de ejecución separado de la página Html desde la que se ha iniciado. De esta manera, el usuario puede manipular esta página, mientras que el script Web Worker trabaja (*work*) en segundo plano.

Los Web Workers permiten ejecutar scripts en paralelo a lo que se muestra en el navegador.



Este API Web Workers se basa en los trabajos del módulo WorkerPool, de Google Gears.

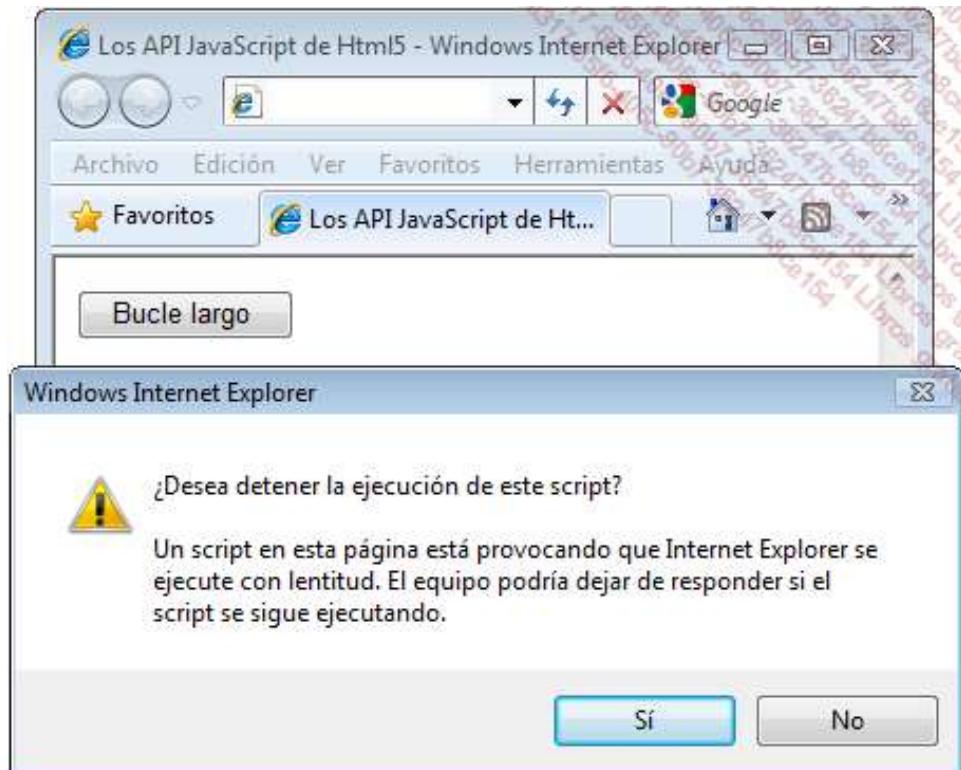
Hay que recordar que los navegadores tienen un sistema de protección quasi histórico para advertir al usuario de que un script consume demasiado tiempo para ejecutarse. Desafortunadamente, este sistema no diferencia los casos donde el script está mal escrito de aquellos que realmente necesitan tiempo para terminar su tratamiento. Después de algunos segundos, el navegador presenta una ventana que propone al usuario detener la ejecución del script.

Por otra parte, la ejecución de una página se realiza de manera secuencial en un flujo único de ejecución. Si un script consume mucho tiempo de ejecución, la página se congela. Esto suele molestar bastante al visitante y se corre el riesgo de que este cierre la pestaña de la página o la instancia del navegador que se está ejecutando.

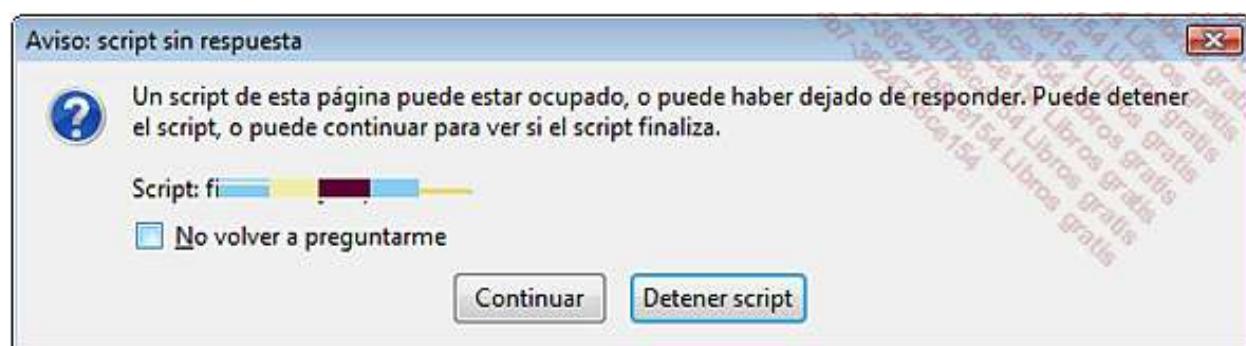
Ejemplo

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<script type="text/javascript">
function bucle(){
for (var i = 0; i <= 10000000000; i += 1){
var j = i;
}
alert("Terminado: " + j + "iteraciones" );
}
</script>
</head>
<body>
<input type="button" onclick="bucle();" value="Bucle largo ">
</body>
</html>
```

Por ejemplo, Internet Explorer muestra el siguiente mensaje después de algunos segundos:



Lo mismo para Firefox:



Este API Web Workers solo ha sido posible después de la aparición en los últimos años de nuevos motores JavaScript en los navegadores más recientes. Google Chrome, con su motor JavaScript Open Source V8, Opera con el proyecto Carakan, Safari en su versión 4 con Nitro o Firefox 3.5 con TraceMonkey, todos intentan mejorar (y algunas veces de manera sensible) el tratamiento de JavaScript; incluso Internet Explorer con su nuevo motor JavaScript Chakra.

Con este API, asociado a los motores JavaScript más potentes de los navegadores, hay procesos cada vez con mejor rendimiento que permiten disminuir la diferencia entre las aplicaciones Web y las aplicaciones de escritorio.

Aunque Google Chrome es capaz de gestionar hasta 16 Workers de manera simultánea, hay que ser consciente a pesar de todo de que estos scripts que se ejecutan en segundo plano consumen recursos (tiempo de ejecución, memoria...) y que no es recomendable ejecutar demasiados al mismo tiempo.

Es importante observar que todos los ejemplos y otras pruebas se tienen que hacer en línea o a través de un servidor local.

Disponibilidad del API

Este API es compatible con los siguientes navegadores de escritorio:

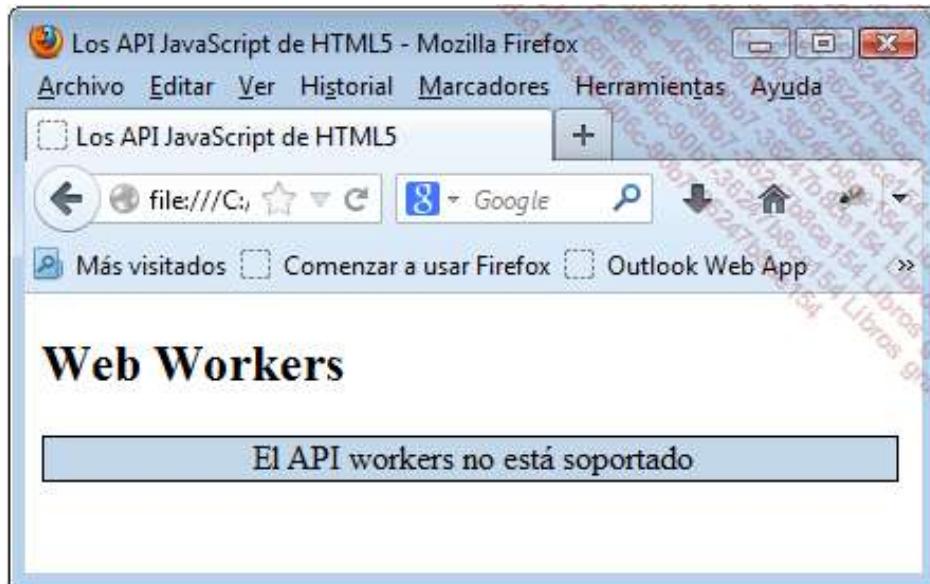
- Internet Explorer 10+
- Firefox 3.6+
- Chrome 4+
- Safari 4+
- Opera 10.6+

Los siguientes navegadores para Smartphone u otras tabletas, también soportan el API:

- iOS Safari 5+
- Android 2.1 (se ha eliminado en las versiones posteriores)
- Opera Móvil 11+

Antes de escribir el código relativo al Web Workers, puede ser útil probar si el navegador reconoce el API.

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#box { width:400px;
        border: 1px solid black;
        background-color: rgb(195,215,235);
        padding-left: 3px;
        text-align: center; }
</style>
<script type="text/javascript">
function init(){
function webworker() {
return (typeof(Worker) !== "undefined") ? true:false;
}
if(webworker() == true) {
msg = "El API Workers no está soportado";
document.querySelector('#box').innerHTML = msg;
}
else {
msg = "El API Workers está soportado";
document.querySelector('#box').innerHTML = msg;
}
}
</script>
</head>
<body onload="init();">
<h2>Web Workers</h2>
<div id="box">&nbsp;</div>
</body>
</html>
```



De manera más concisa:

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#box { border: 1px solid black;
        background-color: rgb(195,215,235);
        padding-left: 3px;
        text-align: center;}
</style>
<script type="text/javascript">
function init(){
if (!!window.Worker) {
msg = "El API Workers está soportado";
document.querySelector('#box').innerHTML = msg;
}
}
</script>
</head>
<body onload="init();">
<h2>Web Workers</h2>
<div id="box">&nbsp;</div>
</body>
</html>
```

Lanzar un script en segundo plano

Un script worker se ejecuta mediante la URL de un archivo JavaScript (separado de la página principal) que contiene el código que el worker debe ejecutar. La sintaxis es muy sencilla:

```
var worker = new Worker("archive_worker.js");
```

Si el archivo existe, el navegador aplica un nuevo entorno en el que el archivo se descarga de manera asíncrona. Si la ruta de acceso a archivo_worker es incorrecta, se produce un error 404 y el worker deja de funcionar de manera silenciosa.

Si su aplicación tiene archivos JavaScript externos, puede utilizar el método `importScripts()` para importarlos en su script worker. Este método recibe diferentes archivos como argumentos, separados por una coma.

```
importScripts("worker1.js, "worker2.js");
```

Ejemplo

Volvamos a nuestro ejemplo del inicio del capítulo, que en JavaScript clásico abriría una ventana preguntando si queremos parar el script.

Con los Web Workers, este bucle largo se realiza sin que esta ventana aparezca, ya que el script se ha lanzado en un entorno de ejecución distinto.

Si echa un vistazo al código, descubrirá que esto no es sencillo, ya que es necesario permitir a la página principal y al worker que trabajen en un hilo de ejecución distinto de comunicaciones. Aquí es donde está la importancia del método `postMessage()`, que se detallará en la sección Comunicación con el script Workers (`postMessage`) de este capítulo.



El archivo Html

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<script type="text/javascript">
var worker = new Worker('bucle.js');
worker.onmessage = function (event) {
alert(event.data + " iteraciones realizadas");
};
</script>
```

```
</head>
<body>
<h2>Web Workers</h2>
</body>
</html>
```

El archivo JavaScript worker (bucle.js)

```
for (var i = 0; i <= 10000000000; i += 1) {
var j = i;
}
postMessage(j);
```

Comentario

La ventana de alerta solo se desencadena después de algún tiempo; es necesario tener paciencia.

Limitaciones de los scripts Workers

Recordemos que los Workers son tareas en segundo plano que se ejecutan en paralelo con el programa JavaScript principal. Para garantizar la seguridad, se ejecutan en un entorno totalmente separado de la página y no tienen acceso directo a esta.

Los Web Workers no tienen acceso:

- Al DOM.
- Al objeto window.
- Al objeto document.
- A la página padre o principal que lanza el worker. De aquí la importancia de `postMessage()`, que se aborda en el siguiente punto.

Los Web Workers solo tienen acceso a un número reducido de funcionalidades del JavaScript clásico:

- El objeto navigator
- EL objeto location (en modo readonly)
- XMLHttpRequest
- El método `importScripts()`
- Los objetos JavaScript Array, Date, Math, String...
- `setTimeout()` y `clearTimeout()`
- `setInterval()` y `clearInterval()`

Comunicación con el script Workers (postMessage)

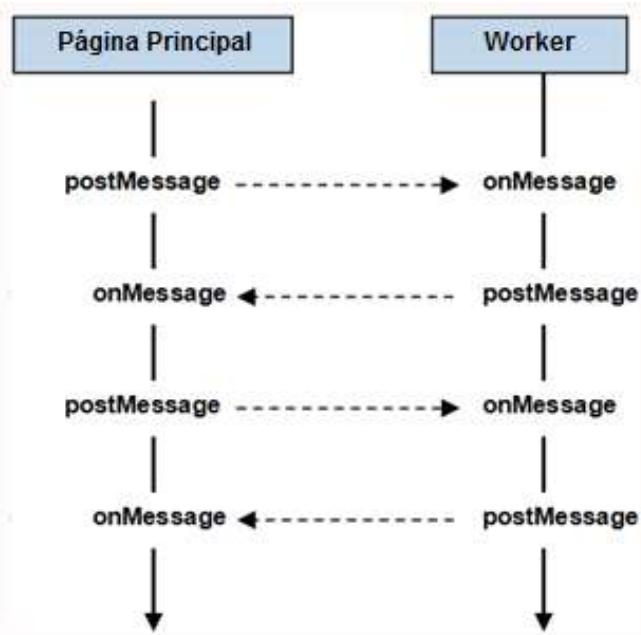
La página Html y el script Workers funcionan en paralelo. Sin embargo, será necesario que se comuniquen entre sí, por ejemplo para enviar al worker los datos o para recibir del worker el resultado de su ejecución. Esta comunicación se realiza mediante los mensajes `postMessage()` para enviar un mensaje, y `onmessage()` durante la recepción de un mensaje. Según los navegadores, los datos de los mensajes se deben proporcionar como argumento en forma de cadena de caracteres o de un objeto JSON.

Para enviar un mensaje, la sintaxis de `postMessage` es:

```
nombre_del_worker.postMessage("dato");
```

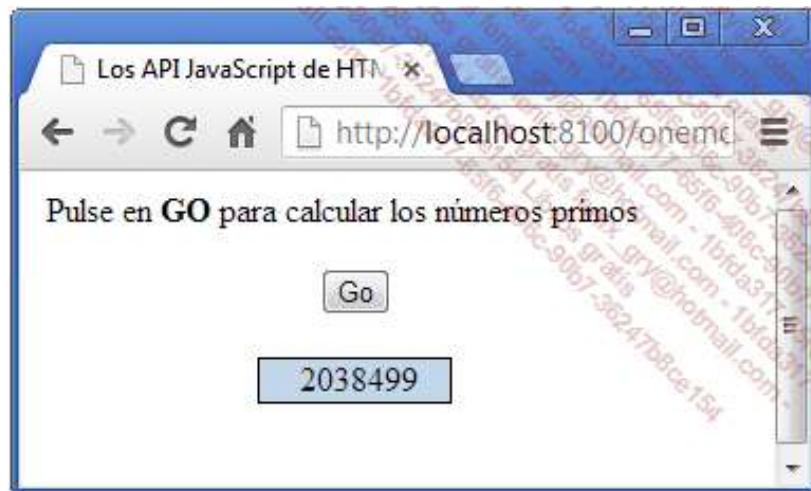
Para recibir un mensaje, es necesario usar el evento `message`. La sintaxis es:

```
nombre_del_worker.onmessage=function(event) {  
...  
};
```



Ejemplo 1

Veamos un ejemplo de cálculo de los números primos por un worker. Este ejemplo es un clásico de la documentación disponible en la Red porque forma parte de la documentación oficial del WHATWG (organismo paralelo al W3C), que se puede consultar en la siguiente dirección:<http://www.whatwg.org/demos/workers/primes/page.html>



El código Html

```
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#box { width: 300px;
        height:100px;
        text-align: center;}
#go { margin-bottom: 20px;}
#out { width: 30%;
        margin: auto;
        border: 1px solid black;
        background-color: rgb(195,215,235);}
</style>
<script type="text/javascript">
function primos() {
var worker = new Worker('primo.js');
worker.onmessage = function (event) {
document.getElementById('result').textContent = event.data;
};
}
</script>
</head>
<body>
<p>Pulse en <b>GO</b> para calcular los números primos</p>
<div id="box">
<input type="button" id="go" value="Go" onClick="primos()">
<p id="out"><output id="result">&nbsp;</output></p>
</div>
</body>
</html>
```

Comentario

```
var worker = new Worker('primo.js');
```

Creación del worker.

```
worker.onmessage = function (event) {
document.getElementById('result').textContent = event.data;
};
```

Cuando la página principal recibe un mensaje del worker, este se muestra en la etiqueta `<output> ... </output>` identificada por result.

El código del worker (prime.js)

```
var n = 1;
search: while (true) {
n += 1;
for (var i = 2; i <= Math.sqrt(n); i += 1)
if (n % i == 0)
continue search;
postMessage(n);
}
```

Comentario

```
postMessage(n)
```

El postMessage del worker devuelve a la página principal la variable n que es el resultado del cálculo de los números primos.

Ejemplo 2

Ilustramos la comunicación entre la página Html y el script Workers.





El código de la página html

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#box { width: 300px;
      border: 1px solid black;
      padding-left: 3px;}
button { margin-top: 18px;
          margin-bottom: 18px;}
</style>
<script type="text/javascript">
hola = new Worker('hola.js');
var f;
function callhola() {
f=document.getElementById("texto")
document.getElementById("holaHtml").innerHTML = "La página HTML
ha dicho: " + f.value;
hola.postMessage(f.value);
}
hola.onmessage = function(event) {
document.getElementById("hola").innerHTML = event.data;
}
</script>
</head>
<body>
<h2>Comunicación con el worker</h2>
Escriba su nombre:<br>
<input type="text" id="texto"><br>
<button onclick="callhola()">Start</button>
<div id="box">
<span id="holaHtml"></span><br>
<span id="hola"></span>
</div>
</body>
</html>
```

Comentario

```
hola = new Worker('hola.js');
```

Creación del worker.

```
function callhola() {
f=document.getElementById("texto")
document.getElementById("holaHtml").innerHTML = "La página HTML ha
dicho: " + f.value;
```

La función callhola() reproduce simplemente en la etiqueta identificada por holaHtml la codificación de la zona de texto, en ese caso el nombre Ángel María.

```
hola.postMessage(f.value);
}
```

Esta codificación de la zona de texto (f.value) se envía al worker hola como argumento de postMessage().

```
hola.onmessage = function(event) {
document.getElementById("hola").innerHTML = event.data;
}
```

Cuando se recibe un mensaje del worker (onmessage), este se muestra en la etiqueta identificada por hola.

El código del worker

```
function hello(v) {
return "El worker ha respondido: Hola " + v;
}
onmessage = function(event) {
postMessage(hello(event.data));
}
Comentario
onmessage = function(event) {
postMessage(hello(event.data));
}
```

Cuando se recibe un mensaje de la página principal (onmessage), el worker devuelve (postMessage) la respuesta que ha generado la función hello().

Terminar un script en segundo plano

Cuando la página principal ha ejecutado un worker y este ha terminado su tarea, se recomienda finalizar explícitamente el worker porque cada worker consume muchos recursos del navegador. Esta es la función del método `terminate()`. La sintaxis es:

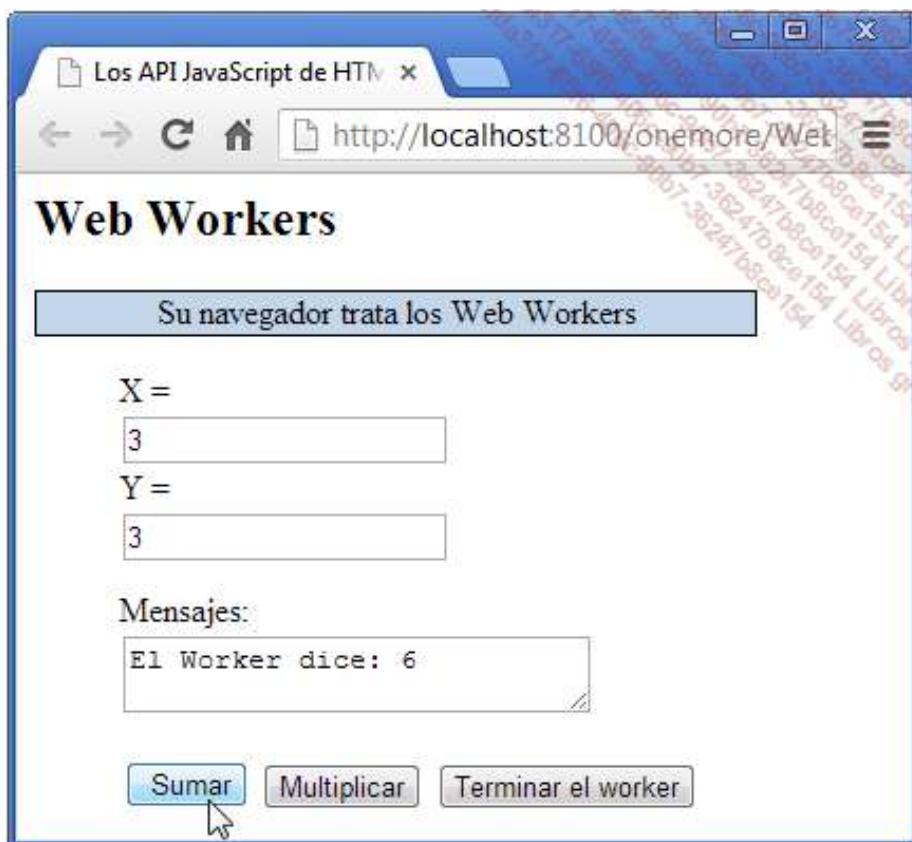
```
nombre_del_worker.terminate();
```

Cuando el worker ha terminado, ya no responde a los mensajes ni ejecutará ninguna operación. No es posible volver a arrancar un worker. En contraposición, puede crear un nuevo worker usando la misma URL.

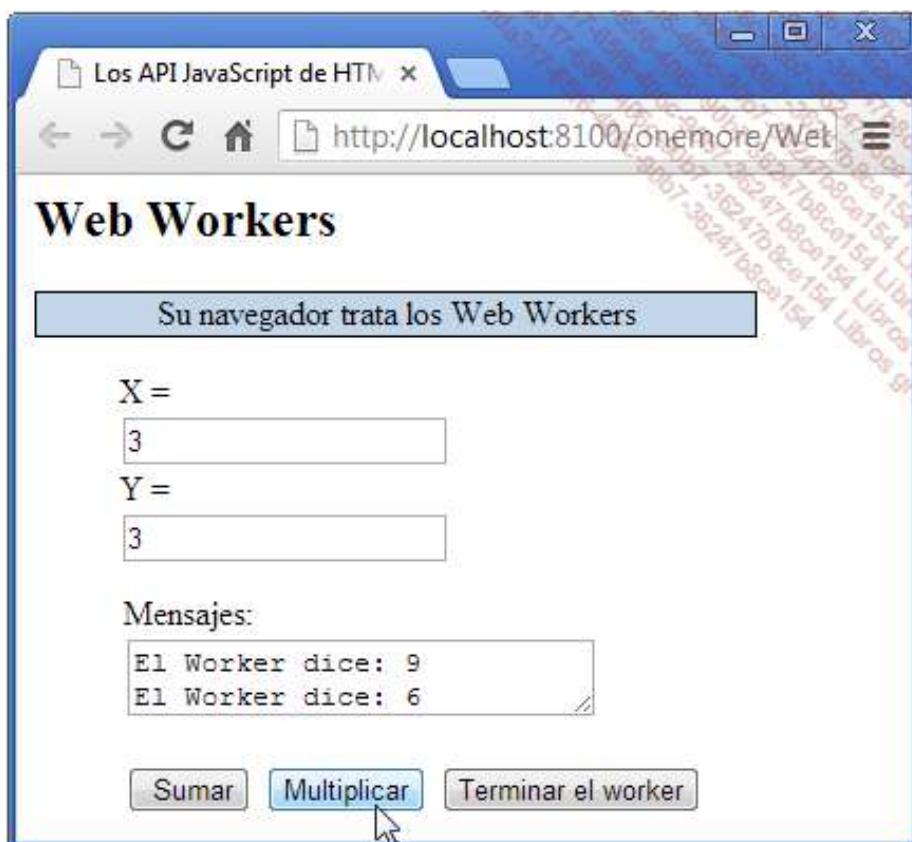
Una aplicación final

Veamos un ejemplo básico de suma y multiplicación, pero que permite recuperar y sintetizar todo lo que hemos visto.

Para la suma:



Para la multiplicación:



El archivo Html

```
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#box { width: 340px;
       border: 1px solid black;
       background-color: rgb(195,215,235);
       text-align: center;}
li { list-style-type:none;}
#li_3 { margin-top: 12px;}
#botones { margin-top: 20px;}
</style>
</head>
<body>
<h2>Web Workers</h2>
<div id="container">
<form id="main">
<p id="box"></p>
<ul>
<li id="li_1">
<label>X =</label>
<div>
<input id="x" name="x" type="text" value="3">
</div>
</li>
<li id="li_2">
<label>Y = </label>
<div>
<input id="y" name="y" type="text" value="3">
</div>
</li>
<li id="li_3">
<label>Mensajes:</label>
<div>
<textarea id="output" name="output" cols="25"></textarea>
</div>
</li>
<li id="botones">
<input id="addButton" type="button" value="Sumar">
<input id="multButton" class="button_text" type="button"
value="Multiplicar">
<input id="killButton" class="button_text" type="button"
value="Terminar el worker">
</li>
</ul>
</form>
</div>
<script type="text/javascript">
var x;
var y;
var message;
function getWebWorkerSupport() {
return (typeof(Worker) !== "undefined")? true:false;
}
if(getWebWorkerSupport() == true)
{
document.getElementById("box").innerHTML = "Su navegador
trata los Web Workers";
calcularWorker = new Worker("calcul.js");
calcularWorker.onmessage = function (event) {
var previousData = document.getElementById("output").value;
```

```

document.getElementById("output").value = 'El worker dice: ' +
event.data + "\n" + previousData;
};

document.getElementById("multButton").onclick = function() {
x = parseFloat(document.getElementById("x").value);
y = parseFloat(document.getElementById("y").value);
message = { 'op': 'mult',
    'x': x,
    'y': y
};
calcularWorker.postMessage(message);
}

document.getElementById("addButton").onclick = function() {
x = parseFloat(document.getElementById("x").value);
y = parseFloat(document.getElementById("y").value);
message = { 'op': 'add' ,
    'x': x,
    'y': y
};
calcularWorker.postMessage(message);
}

document.getElementById("killButton").onclick = function() {
calcularWorker.terminate();
}
}

else {
document.getElementById("box").innerHTML = "Su navegador no
soporta los Web Workers";
document.getElementById("main").enabled = "false";
}
</script>
</body>
</html>

```

Comentario

```

function getWebWorkerSupport() {
return (typeof(Worker) !== "undefined") ? true:false;
}
if(getWebWorkerSupport() == true){
document.getElementById("box").innerHTML = "Su navegador trata
los Web Workers";
}

```

El script verifica si el API Web Worker está soportado por el navegador.

```
calcularWorker = new Worker("calcular.js");
```

Se ha creado un nuevo worker (calcular.js).

```

calcularWorker.onmessage = function (event) {
var previousData = document.getElementById("output").value;
document.getElementById("output").value = 'El worker dice: ' +
event.data + "\n" + previousData;
};

```

Se añade un manejador de eventos al worker, al que se llamará cuando el worker envíe un mensaje.

```
document.getElementById("multButton").onclick = function() {
```

Se guardan los eventos para los botones **Sumar** y **Multiplicar**.

```

x = parseFloat(document.getElementById("x").value);
y = parseFloat(document.getElementById("y").value);
message = { 'op': 'mult',
```

```

        'x': x,
        'y': y
    };
    calcularWorker.postMessage(message);
}

```

Se recuperan los valores para la multiplicación.

```

x = parseFloat(document.getElementById("x").value);
y = parseFloat(document.getElementById("y").value);
message = { 'op': 'add',
            'x': x,
            'y': y
        };
calcularWorker.postMessage(message);
}

```

Se recuperan los valores para la suma.

```

document.getElementById("killButton").onclick = function() {
    calcularWorker.terminate();
}

```

El manejador de eventos para el botón **Terminar el worker**. Nos centraremos especialmente en `calcularWorker.terminate()`, que no habíamos visto en ningún ejemplo.

El archivo worker (calcular.js)

```

function addNumbers(x,y) {
    return x + y;
}
function mulNumbers(x,y) {
    return x*y;
}
this.onmessage = function (event) {
    var dato = event.data;
    switch(data.op) {
        case 'mult':
            postMessage(mulNumbers(data.x, data.y));
            break;
        case 'add':
            postMessage(addNumbers(data.x, data.y));
            break;
        default:
            postMessage("Operación errónea");
    }
};

```

Comentario

```

function addNumbers(x,y) {
    return x + y;
}

```

La función para sumar los números.

```

function mulNumbers(x,y) {
    return x*y;
}

```

La función para multiplicarlos.

```

this.onmessage = function (event) {

```

```
var dato = event.data;
switch(data.op) {
  case 'mult':
    postMessage(mulNumbers(data.x, data.y));
    break;
  case 'add':
    postMessage(addNumbers(data.x, data.y));
    break;
  default:
    postMessage("Operación errónea");
}
};
```

Se añade un manejador de eventos al worker, al que se llama cuando se recibe un mensaje de la página principal.

Presentación, objetivos y preguntas

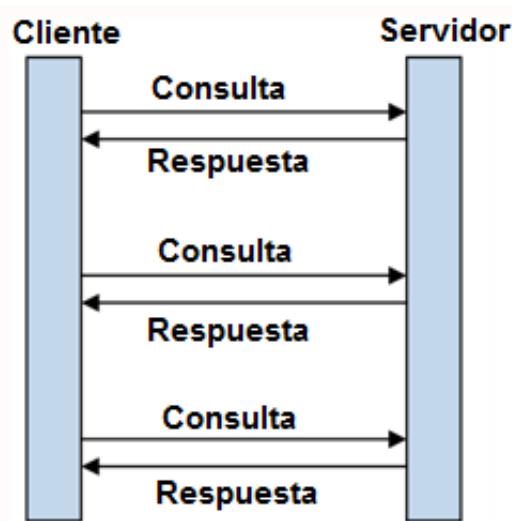
Este API WebSocket permite abrir una conexión bidireccional permanente entre el cliente y el servidor.

El objetivo de este API WebSocket es iniciar un nuevo protocolo en la Red que mantendría un enlace constante entre el navegador y el servidor para disponer, en tiempo real, de información cambiante, como las cotizaciones de la bolsa y los sitios web de información.

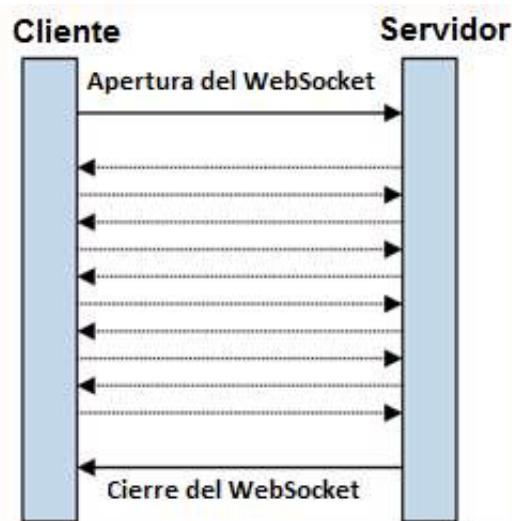
Puede ver una demostración (o una simulación) en la dirección: <http://kaazing.me>

Desde sus inicios, la Red se ha basado en el protocolo de transferencia HTTP (*HyperText Transfer Protocol*). Este protocolo se fundamenta en un sistema de pregunta-respuesta. Es como si durante una entrevista telefónica, fuera necesario recomponer el número de su interlocutor después de cada frase de la conversación.

Con el éxito de Internet, a causa de este procedimiento consulta/respuesta, la carga de trabajo de los servidores es creciente y la adopción masiva de la tecnología Ajax no ha ayudado nada.



El ambicioso proyecto del API WebSocket es guardar la comunicación abierta entre el navegador y el servidor. Esto evita este tiempo de latencia provocado por el procedimiento consulta/respuesta, y permite el desarrollo de aplicaciones todavía más reactivas, es decir en tiempo real.



Este API WebSocket todavía se está desarrollando y se encuentra en estado experimental. Se han descubierto problemas importantes de seguridad, especialmente la posibilidad de corrupción de caché (*cache-poisoning*). Debido a estos problemas de seguridad, algunos navegadores han decidido desactivar el soporte a los WebSockets (cf. sección siguiente).

Pero la especificación es todavía un borrador (*draft*) y el W3C es consciente de estas cuestiones de seguridad y trabaja para solucionarlas. Será necesario esperar algún tiempo para poder utilizar las conexiones WebSocket totalmente seguras. No obstante, como sucede con otros API JavaScript de HTML5, el concepto está en marcha.

Otra de las dificultades es que será necesario tener en cuenta un problema de infraestructura de los servidores WebSocket, que deberán impulsar ellos mismos esta nueva tecnología.

- ▶ Las direcciones de las conexiones WebSocket deben empezar obligatoriamente por ws:// o wss:// para una conexión segura.

Disponibilidad del API

1. Lado cliente

Este API es compatible con los siguientes navegadores de escritorio:

- Internet Explorer 10+
- FireFox 4+, pero está desactivado de manera predeterminada. Se retoma a partir de Firefox 6+.
- Google Chrome 4+
- Safari 5+
- Opera 11+, pero está desactivado de manera predeterminada.

Para los Smartphone y tabletas, esta es la lista de los navegadores compatibles.

- iOS Safari 4.2
- Opera Móvil 11.0
- Android Móvil 2.1

Con semejante embrollo, es más importante que nunca comprobar la compatibilidad con este API antes de empezar a escribir el código.

Una primera posibilidad es dirigirse al sitio Web <http://www.websocket.org/>, que verifica en línea si su navegador soporta o no el protocolo WebSocket. También está el sitio Web <http://jsconsole.com/?WebSocket>, pero este último lo hace de manera menos amigable.

Probémoslos con distintos navegadores.

Sin sorpresa, **Google Chrome** y **Safari** pasan la prueba con éxito.



Este no es el caso de Firefox y Opera.



El caso Firefox

Firefox, después de haber desactivado los WebSockets en su versión 4, los ha retomado a partir de la versión 6+. Pero como sucede a menudo con Firefox, frente a los nuevos estándares no terminados (ver, por ejemplo, con los CSS3), usa los WebSockets con el prefijo moz. Por esta razón, el sitio Web <http://www.websocket.org> no lo reconoce. A nivel del código de los scripts, será necesario usar window.MozWebSocket.

Los lectores interesados en tener más detalles sobre este tema, se pueden dirigir a la página <https://developer.mozilla.org/en/WebSockets>.

El caso Opera

Las operaciones para probar si Opera 11+ soporta el WebSocket son las siguientes:

Abra el navegador Opera.

Escriba opera:config#Enable%20WebSockets en la barra de direcciones. Seleccione **Enable WebSockets**.

Guarde e reinicie Opera.



Para probar la disponibilidad usando código JavaScript, es suficiente con comprobar la presencia del objeto WebSocket en el objeto window. Es decir:

```
if(window.WebSocket) {  
    // El API WebSocket está soportado  
}
```

Para incluir las versiones más recientes de Firefox, el código sería:

```
if(window.WebSocket || window.MozWebSocket) {  
    // El API WebSocket está soportado  
}
```

Ejemplo

```
<!DOCTYPE html>  
<html lang="es">  
<head>  
<title>Los API JavaScript de Html5</title>  
<meta charset=utf-8>  
<style type="text/css">  
#box { width: 200px;
```

```

        border: 1px solid black;
        background-color: rgb(195,215,235);
        padding-left: 3px;
    text-align: center; }
</style>
<script type="text/javascript">
function init(){
if(window.WebSocket) {
msg = "Su navegador soporta el API WebSocket";
document.getElementById('box').innerHTML = msg;
}
else {
msg = "Su navegador no soporta el API WebSocket";
document.getElementById('box').innerHTML = msg;
}
}
</script>
</head>
<body onload="init();">
<h2>API WebSocket</h2>
<div id="box">&nbsp;</div>
</body>
</html>

```



2. Lado servidor

Es necesario un servidor capaz de soportar este nuevo protocolo.

Actualmente existen varias implementaciones de WebSocket del lado servidor:

- Kaazing WebSocket Gateway (<http://kaazing.com/products/Kaazing-websocket-gateway>).
- Jetty (Java).
- Netty (framework Java cliente servidor).
- JWebSocket (Java).
- Web Socket Ruby (Ruby).
- mod_pyWebSocket (extension en Python para el servidor Apache HTTP).
- Websocket (Python).
- ...

Muchos proyectos están en marcha en la actualidad para crear otros servidores y frameworks que

integren los WebSockets.

Para la configuración del servidor, el sitio Web IETF ofrece explicaciones detalladas:<http://tools.ietf.org/html/draft-ietf-hybi-thewebsocketprotocol-06>

- Para nuestros ejemplos, vamos a usar el sitio Web websocket.org, que trata el código de los WebSockets.

De esta manera, usaremos la dirección `ws://echo.websocket.org/`, lo que evitara tener que instalar y configurar un servidor particular para algunos ejemplos.

Los eventos y los métodos

El API WebSocket tiene eventos que permiten gestionar este nuevo tipo de conexión:

onopen	Abre una conexión WebSocket.
onmessage	Recibe un mensaje del servidor.
onerror	Para tratar los errores que se producen durante el proceso.
onclose	Cierra una conexión WebSocket.

Observe que los mensajes enviados por el servidor y notificados por el evento `onmessage` lo son en forma de cadena de caracteres (string).

El API WebSocket proporciona también dos métodos:

<code>send(string)</code>	Permite enviar un mensaje al servidor.
<code>close()</code>	Cierra la conexión WebSocket.

Aplicaciones

Ejemplo 1

Usando dos botones, vamos a abrir y cerrar una conexión WebSocket.

Al iniciar el ejemplo:



Cuando se produce la conexión WebSocket:



Cuando se produce la desconexión:



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#output { width: 200px;
           border: 1px solid black;
           padding-left: 5px;}
#botones { margin-top: 18px;}
</style>
<script language="javascript">
function doConnect() {
var wsUrl;
var output;
output = document.getElementById("output");
testWebSocket();
}
function testWebSocket() {
var wsUrl = "ws://echo.websocket.org/";
websocket = new WebSocket(wsUrl);
websocket.onopen = function(evt) {
onOpen(evt)
};
function onOpen(evt) {
visualizar("Conectado");
}
websocket.onerror = function(evt) {
onError(evt) };
}
function onError(evt) {
visualizar('<span style="color: red;">Error:</span> ' +
evt.data);
}
function doDisconnect(){
visualizar("Desconectado");
websocket.close()
}
function visualizar(message) {
var texto = document.createElement("p");
texto.innerHTML = message;
output.appendChild(texto);
}
```

```

</script>
<body>
<h2>API WebSocket</h2>
<div id="output">&nbsp;</div>
<div id="botones">
<button onclick="doConnect () ">Conectar</button>
<button onclick="doDisconnect () ">Desconectar</button>
</body>
</html>

```

Comentario

```

function testWebSocket() {
...
};

```

La función `testWebSocket()` soporta la apertura de una conexión WebSocket.

```

var wsUrl = "ws://echo.websocket.org/";
websocket = new WebSocket(wsUrl);
websocket.onopen = function(evt) {
onOpen(evt)

```

La variable `wsUrl` contiene la dirección `ws://echo.websocket.org/` utilizada en nuestros ejemplos. El objeto `websocket` instancia una conexión WebSocket con la dirección que se proporciona como argumento (`new WebSocket(wsUrl)`). Después, durante el evento `onopen` relacionado con esta conexión (`websocket.onopen`), el script pasa el control a la función `onOpen(evt)`.

```

function onOpen(evt) {
visualizar("Conectado");
}

```

Cuando se abre la conexión, se muestra un mensaje (`visualizar("Conectado")`).

```

websocket.onerror = function(evt) {
onError(evt) ;
}

```

Si se produce un error relacionado con esta conexión (`websocket.onerror`), el script pasa el control a la función `onError(evt)`.

```

function onError(evt) {
visualizar('<span style="color: red;">Error:</span> ' + evt.data);
}

```

En caso de error, este (`evt.data`) se muestra en rojo.

```

function doDisconnect(){
visualizar("Desconectado");
websocket.close()
}

```

La función `doDisconnect()`, después de haber mostrado un mensaje, interrumpe la conexión, usando para ello el método `close()` (`websocket.close()`).

```

function visualizar(message) {
var texto = document.createElement("p");
texto.innerHTML = message;
output.appendChild(texto);
}

```

La función `visualizar()` gestiona la visualización de los diferentes mensajes.

Ejemplo 2

Vayamos más allá. Además de la apertura y el cierre de una conexión WebSocket, vamos a enviar un mensaje al servidor que nos lo devolverá (ver el echo de la dirección ws://echo.websocket.org/).



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#output { width: 270px;
           border: 1px solid black;
           padding-left: 5px; }
</style>
<script language="javascript">
var output;
function init() {
output = document.getElementById("output");
testWebSocket();
}
function testWebSocket() {
var wsUrl = "ws://echo.websocket.org/";
websocket = new WebSocket(wsUrl);
websocket.onopen = function(evt) {
onOpen(evt)
};
websocket.onclose = function(evt) {
onClose(evt)
};
websocket.onmessage = function(evt) {
onMessage(evt)
};
websocket.onerror = function(evt) {
onError(evt) };
}
function onOpen(evt) {
```

```

visualizar("Conectado");
doSend("Hola a todos ");
}
function onClose(evt) {
visualizar("Desconectado");
}
function onMessage(evt) {
visualizar('<span>Respuesta del servidor: ' + evt.data+'</span>');
websocket.close();
}
function onError(evt) {
visualizar('<span style="color: red;">Error:</span> ' +
evt.data);
}
function doSend(message) {
visualizar("Enviado: " + message);
websocket.send(message);
}
function visualizar(message) {
var texto = document.createElement("p");
texto.innerHTML = message;
output.appendChild(texto);
}
window.addEventListener("load", init, false);
</script>
<body>
<h2>API WebSocket</h2>
Console
<div id="output">&nbsp;</div>
</body>
</html>

```

Comentario

```

function testWebSocket() {
...
};

```

La función `testWebSocket()` realiza la apertura de la conexión WebSocket.

```

var wsUrl = "ws://echo.websocket.org/";
websocket = new WebSocket(wsUrl);
websocket.onopen = function(evt) {
onOpen(evt)

```

La variable `wsUrl` contiene la dirección `ws://echo.websocket.org/`, que usamos en nuestros ejemplos. El objeto `websocket` instancia una conexión WebSocket con la dirección que se proporciona como argumento (`new WebSocket(wsUrl)`). Después, durante el evento `onopen` relacionado con esta conexión (`websocket.onopen`), el script cede el control a la función `onOpen(evt)`.

```

websocket.onclose = function(evt) {
onClose(evt)
};
websocket.onmessage = function(evt) {
onMessage(evt)
};
websocket.onerror = function(evt) {
onError(evt) };
}
```

El script define, para una serie de eventos (`onclose`, `onmessage`, `onerror`), las funciones que se les asocian (`onClose(evt)`, `onMessage(evt)` y `onError(evt)`).

```
function onOpen(evt) {
```

```
visualizar("Conectado");
doSend("Hola a todos ");
}
```

La función `onOpen(evt)` pide a la función `visualizar()` que confirme la conexión Websocket y prepara un mensaje que se envía al servidor usando la función `doSend()`.

```
function onClose(evt) {
visualizar("Desconectado");
}
```

La función `onClose(evt)` mostrará un mensaje de fin de conexión.

```
function onMessage(evt) {
visualizar('<span>Respuesta del servidor: ' + evt.data+'</span>');
websocket.close();
}
```

La función `onMessage(evt)` mostrará en la consola la respuesta del servidor. Recordemos que esta será idéntica al mensaje enviado. Entonces el script cierra directamente la conexión WebSocket usando el método `websocket.close()`.

```
function onError(evt) {
visualizar('<span style="color: red;">Error:</span> ' + evt.data);
}
```

La función `onError(evt)` gestiona los errores de conexión que se puedan producir y muestra el tipo (`evt.data`).

```
function doSend(message) {
visualizar("Enviado: " + message);
websocket.send(message);
}
```

La función `doSend(message)` muestra en la consola el mensaje del usuario, preparado antes en el script y lo envía con el método `send()` al servidor (`websocket.send(message)`).

```
function visualizar(message) {
var texto = document.createElement("p");
texto.innerHTML = message;
output.appendChild(texto);
}
```

Muestra en la consola los diferentes mensajes que genera el script, con la función `visualizar(message)`.

```
window.addEventListener("load", init, false);
```

Lanza la función `init()` al cargar la página (`load`).

Presentación y objetivo

La nueva etiqueta <canvas> de Html5 permite integrar las zonas de diseño en 2D en el código de la página. Estos diseños realizados gracias al API Canvas permiten, por ejemplo, visualizar gráficos de Excel sin pasar por capturas de pantalla. Por otra parte, estos elementos podrían ser dinámicos utilizando scripts JavaScript.

El API Canvas deja entrever la cantidad de oportunidades en términos de diseño y grafismo, sobre todo en lo relativo al diseño 3D. Algunos no dudan en verlo como un competidor serio de Flash.

Disponibilidad del API

Este API está disponible para los siguientes navegadores de escritorio:

- Internet Explorer 9+
- Firefox 3.6+
- Safari 5+
- Google Chrome 7+
- Opera 10.6+

Para los Smartphone y tabletas, esta es la lista de navegadores compatibles:

- iOS Safari 3.2
- Opera Móvil 9.0
- Android Móvil 2.1

Podemos concluir que el API Canvas está bastante extendido entre los navegadores recientes.

Para probar el soporte del API Canvas, comprobamos el valor true o false del siguiente código:

```
(!!document.createElement('canvas').getContext)
```

De hecho, si el navegador soporta el API Canvas, el objeto Canvas que genera tendrá el método específico `getContext()`. Entonces creamos en JavaScript un objeto de diseño Canvas ficticio con `document.createElement('canvas')` al que vamos a aplicar el método `getContext()`. Para terminar, el doble operador negativo `!!` fuerza el resultado a tomar un valor booleano (true o false).

Ejemplo

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#box { width: 300px;
        border: 1px solid black;
        background-color: rgb(195,215,235);
        text-align: center;}
</style>
<script type="text/javascript">
function init() {
if (!!document.createElement('canvas').getContext) {
msg = "El API Canvas está soportado";
document.querySelector('#box').innerHTML = msg;
}
else {
msg = "El API Canvas no está soportado";
document.querySelector('#box').innerHTML = msg;
}
}
window.onload = function() {
init();
}
</script>
</head>
<body>
<h2>El API Canvas</h2>
```

```
<div id="box">&nbsp;</div>
</body>
</html>
```



Definir la zona de diseño

En primer lugar, la zona de diseño del API Canvas se define con la etiqueta Html5 <canvas>. Es dentro de esta zona donde se aplican el diseño y otros grafismos.

Las dimensiones de esta zona se introducen con los atributos width y height, que fijan la anchura y altura. Por ejemplo:

```
<canvas width="200px" height="200px"></canvas>
```

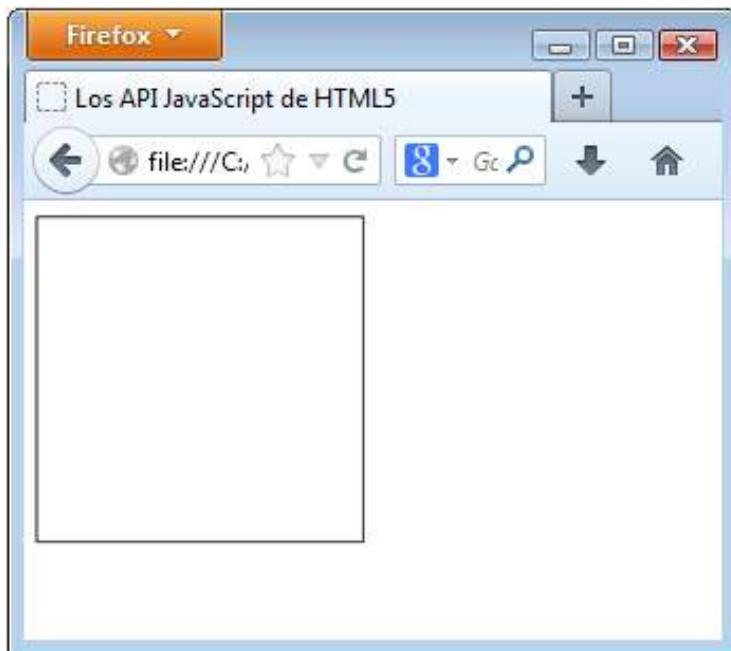
Por defecto, esta zona de representación es transparente. Una captura de pantalla en este momento es inútil.

Para visualizar esta zona introducida por la etiqueta <canvas>, añadimos con una propiedad de estilo un sencillo borde de 1 px.

Por otra parte, prevemos un identificador de tipo id para poder utilizar en JavaScript esta zona gráfica iniciada con la etiqueta <canvas>.

El código se convierte en:

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#zone { border: 1px solid black; }
</style>
</head>
<body>
<canvas id="zone" width="200px" height="200px">
</canvas>
</body>
</html>
```



La etiqueta <canvas> es compatible con las versiones más recientes de los navegadores. Sin embargo, sigue siendo prudente prever un contenido alternativo para los navegadores más antiguos, que no la soportan.

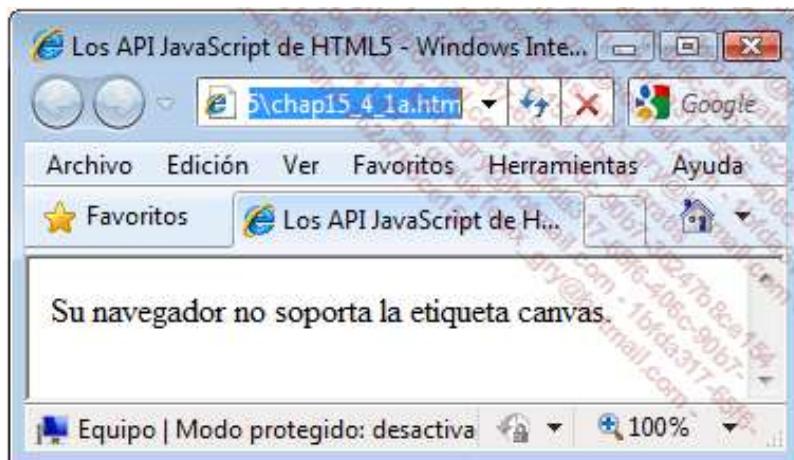
```
<canvas id="zone" width="200px" height="200px">
```

```
Su navegador no soporta la etiqueta canvas.  
</canvas>
```

El código completo es:

```
<!DOCTYPE html>  
<html lang="es">  
<head>  
<title>Los API JavaScript de Html5</title>  
<meta charset=utf-8>  
<style type="text/css">  
#zone { border: 1px solid black; }  
</style>  
</head>  
<body>  
<canvas id="zone" width="200px" height="200px">  
Su navegador no soporta la etiqueta canvas.  
</canvas>  
</body>  
</html>
```

Este es el resultado en Internet Explorer 8, que no reconoce la etiqueta <canvas>.



Comentario

- Cuando los atributos width y height no se especifican, el lienzo tendrá por defecto una anchura de 300 píxeles y una altura de 150 píxeles.
- Es posible tener varias etiquetas <canvas> en la misma página. Es en este caso cuando el identificador muestra toda su utilidad.
- El elemento <canvas> se puede estilizar de la misma manera que cualquier imagen normal (márgenes, bordes, color de fondo, etc.). Estas reglas no afectan en nada al diseño del canvas en sí mismo.

Y todo para Html5. La etiqueta <canvas> solo muestra una zona vacía. Antes de empezar el proceso de diseño o grafismo con el API Canvas, hay que definir el contexto de renderizado del método getContext(), que toma como argumento el tipo de contexto, es decir, el diseño 2D en el marco de este libro.

El código

```
<!DOCTYPE html>  
<html lang="es">  
<head>  
<title>Los API JavaScript de Html5</title>  
<meta charset=utf-8>
```

```
<style type="text/css">
#zone { border: 1px solid black; }
</style>
</head>
<body>
<canvas id="zone" width="200px" height="200px">
Su navegador no soporta la etiqueta canvas.
</canvas>
<script type="text/javascript">
var miCanvas = document.getElementById("zone");
var contexto = miCanvas.getContext("2d");
// Continuación del script
</script>
</body>
</html>
```

Detallamos estas nuevas líneas de código.

```
var miCanvas = document.getElementById("zone");
```

La variable **miCanvas** contiene la zona de diseño del documento identificada por **zone**.

```
var contexto = miCanvas.getContext("2d");
```

La variable **contexto** va a buscar el API Canvas de diseño 2D (`getContext ("2d")`) y lo aplica a la variable **miCanvas**.

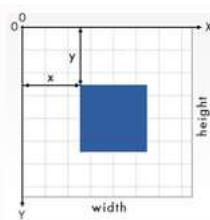
Estamos preparados para iniciar las funcionalidades de diseño de este API.

 Las especificaciones de Html5 prevén para el futuro diseños en 3D. Las aplicaciones, en estado experimental, funcionan ya en Opera 10, pero todavía estamos un poco lejos de una estandarización.

Las formas geométricas

1. El rectángulo

Antes de pasar al código relativo a los diseños, es necesario especificar la notación que se usa para definir las coordenadas.



El origen de los ejes x e y se ubica en la esquina superior izquierda [coordenadas (0,0)]. Todos los elementos del canvas se definen respecto a este punto de origen. De esta manera, el borde superior izquierdo de la zona coloreada se define a x píxeles horizontalmente desde la izquierda e y píxeles verticalmente desde el borde superior [coordenadas (x,y)].

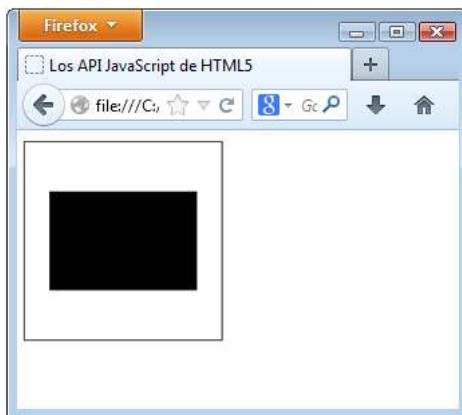
Un rectángulo completo se diseña con la función:

```
fillRect(x,y,anchura,altura)
```

Ejemplo

Diseñamos un rectángulo completo cuyo borde superior izquierdo está situado a 25 píxeles del borde izquierdo y a 50 píxeles del borde superior. Nuestro rectángulo tiene una anchura (width) de 150 píxeles y una altura (height) de 100 píxeles. Observe que por defecto el color es negro. Más adelante en este capítulo veremos cómo dar color a los diseños.

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#zone { border: 1px solid black; }
</style>
</head>
<body>
<canvas id="zone" width="200px" height="200px">
Su navegador no soporta la etiqueta canvas.
</canvas>
<script type="text/javascript">
var miCanvas = document.getElementById("zone");
var contexto = miCanvas.getContext("2d");
contexto.fillRect(25,50,150,100);
</script>
</body>
</html>
```

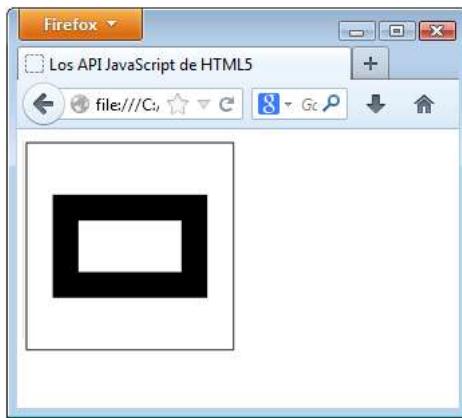


La función `clearRect(x,y,anchura,altura)` borra una zona especificada y la hace totalmente transparente.

Cortemos una zona en el rectángulo que hemos definido antes.

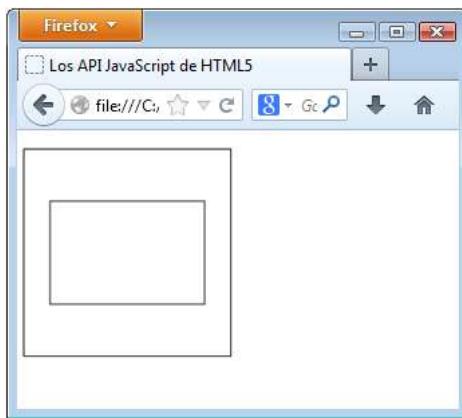
```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#zone { border: 1px solid black; }
</style>
</head>
<body>
<canvas id="zone" width="200px" height="200px">
Su navegador no soporta la etiqueta canvas.
</canvas>
<script type="text/javascript">
var miCanvas = document.getElementById("zone");
var contexto = miCanvas.getContext("2d");
contexto.fillRect(25,50,150,100);
contexto.clearRect(50,75,100,50);
</script>
</body>
</html>
```

```
</script>
</body>
</html>
```



Para terminar, la función `strokeRect(x, y, anchura, altura)` diseña un rectángulo vacío que solo tiene el borde visible.

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de HTML5</title>
<meta charset=utf-8>
<style type="text/css">
#zone { border: 1px solid black; }
</style>
</head>
<body>
<canvas id="zone" width="200px" height="200px">
Su navegador no soporta la etiqueta canvas.
</canvas>
<script type="text/javascript">
var miCanvas = document.getElementById("zone");
var contexto = miCanvas.getContext("2d");
contexto.strokeRect(25,50,150,100);
</script>
</body>
</html>
```



2. Añadir color

Para añadir color tiene las propiedades `fillStyle="color"` y `strokeStyle="color"`.

La propiedad `fillStyle` se usa para el color de relleno de las formas geométricas y `strokeStyle` para definir el color del contorno de la forma y las líneas. Por defecto, los colores del contorno y de relleno se definen en negro.

La notación de los colores acepta todas las especificaciones de color CSS, incluso las de CSS3. De esta manera, el color naranja se puede definir indiferentemente por:

- `ctx.fillStyle = "orange";`
- `ctx.fillStyle = "#FFA500";`
- `ctx.fillStyle = "rgb(255,165,0)";`
- `ctx.fillStyle = "rgba(255,165,0,1)";`

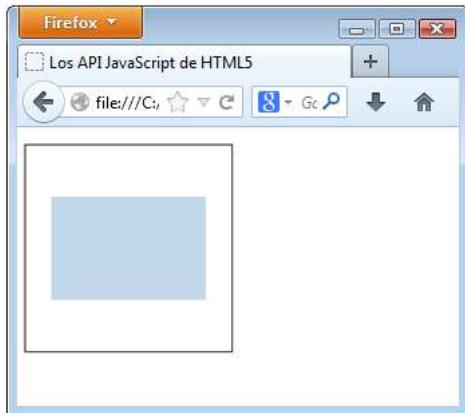
Ejemplo con `fillStyle`

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de HTML5</title>
<meta charset=utf-8>
<style type="text/css">
#zone { border: 1px solid black; }
</style>
</head>
```

```

<body>
<canvas id="zone" width="200px" height="200px">
Su navegador no soporta la etiqueta canvas.
</canvas>
<script type="text/javascript">
var miCanvas = document.getElementById("zone");
var contexto = miCanvas.getContext("2d");
contexto.fillStyle = "rgb(195,215,235)";
contexto.fillRect(25,50,150,100);
</script>
</body>
</html>

```



3. La línea recta

El diseño de una línea siempre empieza por la función `beginPath()` por motivos de gestión interna del navegador. Las líneas, arcos, etc., se almacenan en una lista y la función opcional `closePath()` indica que la construcción ha terminado.

El método `lineTo(x, y)` permite diseñar líneas rectas. Los argumentos `x` e `y` son las coordenadas de los extremos de la línea. El punto inicial depende de los trazos diseñados anteriormente, ya que el último punto de un trazo es el punto inicial del siguiente, etc. El punto inicial también se puede desplazar con la ayuda del método `moveTo(x, y)`.

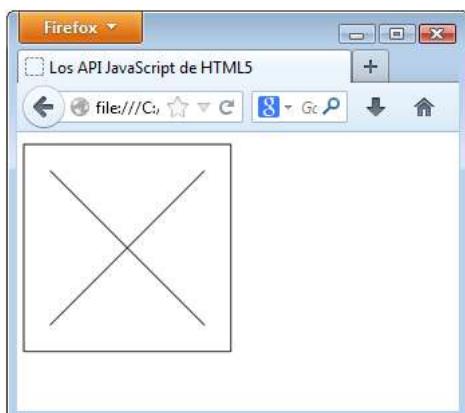
- ▶ Para entender el API Canvas, hay que imaginar que diseña con la ayuda de un brazo articulado provisto de un lápiz. Utilizando las funciones y los métodos del API Canvas, puede dirigir este brazo para construir el gráfico que deseé.

Ejemplo

```

<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#zone { border: 1px solid black; }
</style>
</head>
<body>
<canvas id="zone" width="200px" height="200px">
Su navegador no soporta la etiqueta canvas.
</canvas>
<script type="text/javascript">
var miCanvas = document.getElementById("zone");
var contexto = miCanvas.getContext("2d");
contexto.beginPath();
contexto.moveTo(25,25);
contexto.lineTo(175,175);
contexto.moveTo(175,25);
contexto.lineTo(25,175);
contexto.stroke();
</script>
</body>
</html>

```



Detallemos el código:

```
contexto.moveTo(25,25);
contexto.lineTo(175,175);
```

El script coloca el lápiz virtual en el punto inicial de la línea (`moveTo(25,25)`) y diseña la primera diagonal (`lineTo(175,175)`).

```
contexto.moveTo(175,25);
contexto.lineTo(25,175);
```

Después, el lápiz se mueve hasta el extremo de la segunda diagonal (`moveTo(175,25)`) y la traza (`lineTo(25,175)`).

```
contexto.stroke();
```

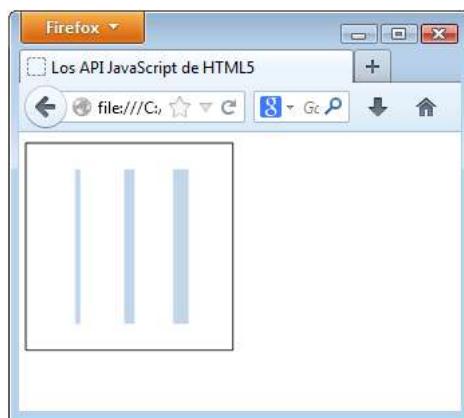
Con los rectángulos, todas las operaciones (diseño, color, etc.) se aplican directamente sobre el lienzo. Para el resto de las formas, usa un lápiz (virtual) sin tinta. De alguna manera, es necesario hacer aparecer la tinta al final del diseño, de ahí el código `stroke()`.

Hay varias propiedades que permiten cambiar el estilo de las líneas.

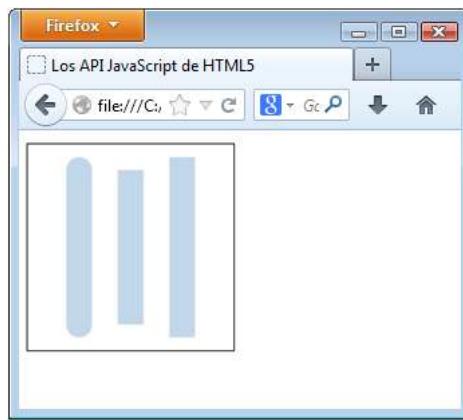
La propiedad `lineWidth = valor` define el ancho de la línea actual. El valor por defecto es 1 píxel.

Vamos a diseñar varias líneas verticales con diferentes anchos.

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#zone { border: 1px solid black; }
</style>
</head>
<body>
<canvas id="zone" width="200px" height="200px">
Su navegador no soporta la etiqueta canvas.
</canvas>
<script type="text/javascript">
var miCanvas = document.getElementById("zone");
var contexto = miCanvas.getContext("2d");
contexto.strokeStyle = "rgb(195,215,235)";
contexto.beginPath();
contexto.lineWidth = 5;
contexto.moveTo(50,25);
contexto.lineTo(50,175);
contexto.stroke();
contexto.beginPath();
contexto.moveTo(100,25);
contexto.lineWidth = 10;
contexto.lineTo(100,175);
contexto.stroke();
contexto.beginPath();
contexto.moveTo(150,25);
contexto.lineWidth = 15;
contexto.lineTo(150,175);
contexto.stroke();
</script>
</body>
</html>
```



La propiedad `lineCap = valor` determina cómo se diseñan los extremos de cada línea. Los tres valores posibles para esta propiedad son: `butt` (valor predeterminado), `round` y `square`.



La captura de pantalla y el siguiente ejemplo ilustra cada uno de los valores de la propiedad `lineCap`:

- El valor `round` añade un semicírculo al extremo de la línea. Su radio es la mitad del ancho de la línea.
- El valor `butt` (valor predeterminado) termina exactamente en el lugar definido por el código.
- El valor `square` añade una caja de la anchura de la línea y con una altura la mitad del ancho de esta.

Ejemplo

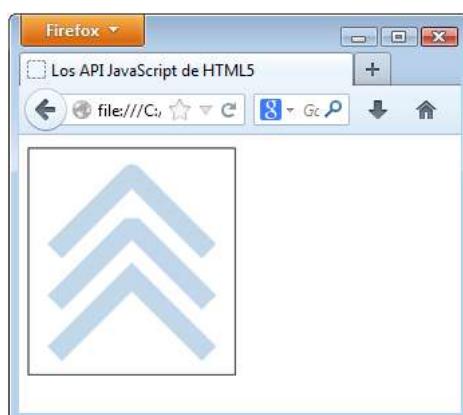
```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de HTML5</title>
<meta charset="utf-8">
<style type="text/css">
#zone { border: 1px solid black; }
</style>
</head>
<body>
<canvas id="zone" width="200px" height="200px">
Su navegador no soporta la etiqueta canvas.
</canvas>
<script type="text/javascript">
var miCanvas = document.getElementById("zone");
var contexto = miCanvas.getContext("2d");
contexto.strokeStyle = "rgb(195,215,235)";
contexto.lineWidth = 25;
contexto.beginPath();
contexto.moveTo(50,25);
contexto.lineCap="round";
contexto.lineTo(50,175);
contexto.stroke();
contexto.beginPath();
contexto.lineCap="butt";
contexto.moveTo(100,25);
contexto.lineTo(100,175);
contexto.stroke();
contexto.beginPath();
contexto.moveTo(150,25);
contexto.lineCap="square";
contexto.lineTo(150,175);
contexto.stroke();
</script>
</body>
</html>
```

La propiedad `lineJoin=valor` define la manera en que dos líneas que se unen se conectan entre ellas. Los tres valores posibles para esta propiedad son: `round`, `bevel` y `miter`. Por defecto, el valor que se utiliza es `miter`.

La captura de pantalla y el siguiente ejemplo ilustran cada uno de los valores de la propiedad `lineJoin`:

El valor `round` redondea las esquinas de la forma. El radio del redondeo es igual al ancho de la línea.

- El valor `bevel` no hace ninguna operación.
- El valor `miter` (valor predeterminado) alarga las líneas con las que se une en un solo punto.



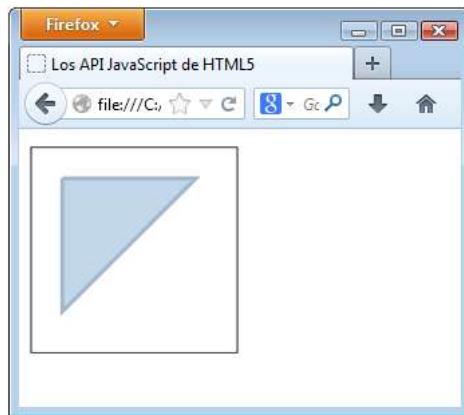
Ejemplo

```

<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#zone { border: 1px solid black; }
</style>
</head>
<body>
<canvas id="zone" width="200px" height="220px">
Su navegador no soporta la etiqueta canvas.
</canvas>
<script type="text/javascript">
var miCanvas = document.getElementById("zone");
var contexto = miCanvas.getContext("2d");
contexto.strokeStyle = "rgb(195,215,235)";
contexto.lineWidth = 20;
contexto.beginPath();
contexto.lineJoin="round";
contexto.moveTo(25,100);
contexto.lineTo(100,25);
contexto.lineTo(175,100);
contexto.stroke();
contexto.beginPath();
contexto.lineJoin="bevel";
contexto.moveTo(25,150);
contexto.lineTo(100,75);
contexto.lineTo(175,150);
contexto.stroke();
contexto.beginPath();
contexto.lineJoin="miter";
contexto.moveTo(25,200);
contexto.lineTo(100,125);
contexto.lineTo(175,200);
contexto.stroke();
</script>
</body>
</html>

```

El API Canvas no solo permite diseñar rectángulos. A continuación se muestra un ejemplo de diseño de un triángulo.



El código

```

<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#zone { border: 1px solid black; }
</style>
</head>
<body>
<canvas id="zone" width="200px" height="200px">
Su navegador no soporta la etiqueta canvas.
</canvas>
<script type="text/javascript">
var miCanvas = document.getElementById("zone");
var contexto = miCanvas.getContext("2d");
contexto.fillStyle = "rgb(195,215,235)";
contexto.strokeStyle = "rgb(165,185,205)";
contexto.lineWidth = 4;
contexto.beginPath();
contexto.moveTo(30,30);
contexto.lineTo(160,30);
contexto.lineTo(30,160);
contexto.lineTo(30,30);
contexto.fill();
contexto.stroke();
</script>
</body>
</html>

```

4. Los arcos, círculos y otras formas

Para diseñar arcos y círculos, hay que utilizar el método `arc(x, y, radio, ánguloInicio, ánguloFin, sentidoInverso)`.

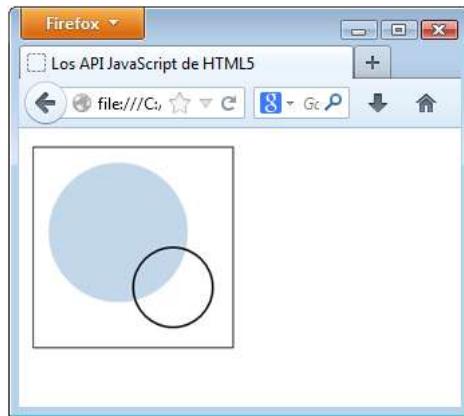
Este método tiene cinco argumentos:

- `x` e `y` son las coordenadas del centro del círculo.

- radio es su radio.
- Los argumentos `ánguloInicio` y `ánguloFin` definen los puntos de inicio y fin del arco en radianes. Son medidas tomadas respecto al eje x.
- El argumento `sentidoInverso` es un valor booleano que, cuando vale `true`, diseña el arco en sentido inverso a las agujas del reloj. El valor `false` diseña el arco en el sentido de las agujas del reloj.

Recordemos que los ángulos se expresan en radianes, y no en grados.

Ejemplo



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#zone { border: 1px solid black; }
</style>
</head>
<body>
<canvas id="zone" width="200px" height="200px">
Su navegador no soporta la etiqueta canvas.
</canvas>
<script type="text/javascript">
var miCanvas = document.getElementById("zone");
var contexto = miCanvas.getContext("2d");
contexto.fillStyle = "rgb(195,215,235)";
contexto.lineWidth = 2;
contexto.beginPath();
contexto.arc(85, 85, 70, 0, 2*Math.PI, true);
contexto.fill();
contexto.beginPath();
contexto.strokeStyle = "black";
contexto.arc(140, 140, 40, 0, 2*Math.PI, true);
contexto.stroke();
</script>
</body>
</html>
```

Para las formas complejas, las curvas de Bézier también están disponibles en el API Canvas. Las curvas de Bézier diseñan curvas a partir de fórmulas matemáticas. Su aprendizaje es más complejo que el de las curvas e implica una formación importante en infografía. Se mencionan a título informativo.

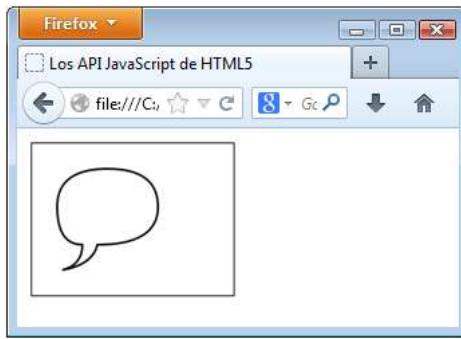
Hay dos funciones disponibles, `quadraticCurveTo` y `bezierCurveTo`. Una curva de Bézier cuadrática tiene un punto inicial, un punto final y un punto único de control. Una curva de Bézier cúbica tiene dos puntos de control.

El uso de las curvas de Bézier cuadráticas y cúbicas puede resultar bastante arduo, ya que, al contrario de lo que sucede con el software de diseño vectorial como Adobe Illustrator, no tenemos un ejemplo visual directo de lo que estamos diseñando, lo que no facilita la creación de formas complejas.

Veamos algunos ejemplos sencillos.

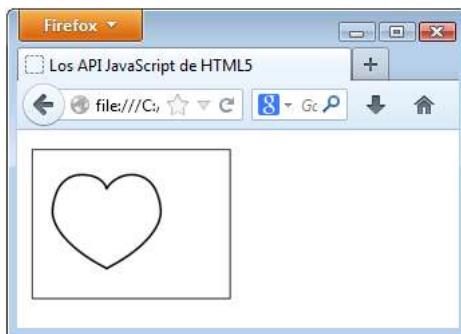
El primero usa la función `quadraticCurveTo`.

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#zone { border: 1px solid black; }
</style>
</head>
<body>
<canvas id="zone" width="200px" height="150px">
Su navegador no soporta la etiqueta canvas.
</canvas>
<script type="text/javascript">
var miCanvas = document.getElementById("zone");
var contexto = miCanvas.getContext("2d");
contexto.beginPath();
contexto.lineWidth = 2;
contexto.moveTo(75,25);
contexto.quadraticCurveTo(25,25,25,62.5);
contexto.quadraticCurveTo(25,100,50,100);
contexto.quadraticCurveTo(50,120,30,125);
contexto.quadraticCurveTo(60,120,65,100);
contexto.quadraticCurveTo(125,100,125,62.5);
contexto.quadraticCurveTo(125,25,75,25);
contexto.stroke();
</script>
</body>
</html>
```



El segundo usa bezierCurveTo.

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#zone { border: 1px solid black; }
</style>
</head>
<body>
<canvas id="zone" width="200px" height="150px">
Su navegador no soporta la etiqueta canvas.
</canvas>
<script type="text/javascript">
var miCanvas = document.getElementById("zone");
var contexto = miCanvas.getContext("2d");
contexto.beginPath();
contexto.lineWidth = 2;
contexto.moveTo(75,40);
contexto.bezierCurveTo(75,37,70,25,50,25);
contexto.bezierCurveTo(20,25,20,62.5,20,62.5);
contexto.bezierCurveTo(20,80,40,102,75,120);
contexto.bezierCurveTo(110,102,130,80,130,62.5);
contexto.bezierCurveTo(130,62.5,130,25,100,25);
contexto.bezierCurveTo(85,25,75,37,75,40);
contexto.stroke();
</script>
</body>
</html>
```



5. Añadir un degradado de color

Como sucede en cualquier aplicación gráfica, es posible llenar las formas con degradados de color lineales o radiales. Para ello, creamos un objeto `canvasGradient` con uno de los siguientes métodos:

```
createLinearGradient(x1, y1, x2, y2)
para los degradados lineales (Linear).

createRadialGradient(x1, y1, r1, x2, y2, r2)
para los degradados radiales (Radial).
```

El método `createLinearGradient` tiene cuatro argumentos que representan el punto inicial (x_1, y_1) y final (x_2, y_2) del degradado.

Por ejemplo:

```
var lineargradient = new contexto.createLinearGradient(0, 0, 150, 150);
```

El método `createRadialGradient` tiene seis argumentos. Los tres primeros definen un círculo centrado en las coordenadas (x_1, y_1) y de radio r_1 , y los siguientes, un círculo centrado en las coordenadas (x_2, y_2) y de radio r_2 .

Por ejemplo:

```
var radialgradient = new contexto.createRadialGradient(75, 75, 0, 75, 75, 100);
```

Una vez creado este objeto `canvasGradient`, le podemos asignar colores usando el método `addColorStop`.

```
addColorStop(position, color)
```

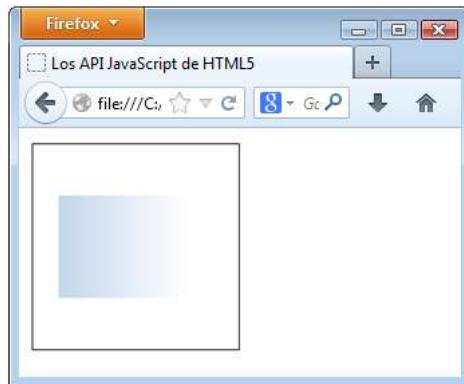
Este método tiene dos argumentos:

- El argumento position, que debe ser un número entre 0.0 y 1.0 y define la posición relativa del color en el degradado. Por ejemplo, situándolo a 0.5, ubicamos el color justo en el centro del degradado.
- El argumento color debe ser una cadena de caracteres que represente un color CSS.

Para terminar, asignamos a este objeto las propiedades fillStyle o strokeStyle.

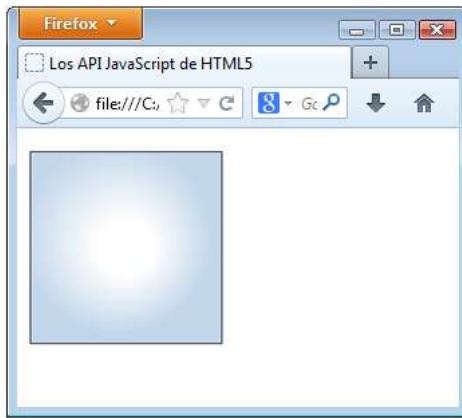
Ejemplo de gradiente lineal

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#zone { border: 1px solid black; }
</style>
</head>
<body>
<canvas id="zone" width="200px" height="200px">
Su navegador no soporta la etiqueta canvas.
</canvas>
<script type="text/javascript">
var miCanvas = document.getElementById("zone");
var contexto = miCanvas.getContext("2d");
var gradienteLinear = contexto.createLinearGradient(25,87,150,87);
gradienteLinear.addColorStop(0,'rgb(195,215,235)');
gradienteLinear.addColorStop(1,'#fff');
contexto.fillStyle = gradienteLinear;
contexto.fillRect(25,50,150,100);
</script>
</body>
</html>
```



Ejemplo de gradiente radial

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#zone { border: 1px solid black; }
</style>
</head>
<body>
<canvas id="zone" width="200px" height="200px">
Su navegador no soporta la etiqueta canvas.
</canvas>
<script type="text/javascript">
var miCanvas = document.getElementById("zone");
var contexto = miCanvas.getContext("2d");
var radgrad = contexto.createRadialGradient(100,100,30,90,90,100);
radgrad.addColorStop(0,'white');
radgrad.addColorStop(1,'rgb(195,215,235)');
contexto.fillStyle = radgrad;
contexto.fillRect(0,0,200,200);
</script>
</body>
</html>
```



6. Añadir transparencias

Además de diseñar formas opacas en el lienzo, también es posible aportar transparencias a los colores. Esto se realiza con la propiedad `globalAlpha`.

```
globalAlpha = valor_de_transparencia
```

donde el valor de transparencia está comprendido entre 0.0 (totalmente transparente) y 1.0 (totalmente opaco).

Gracias a la notación de colores CSS3, podemos añadir fácilmente una transparencia utilizando, para `strokeStyle` y `fillStyle`, la notación `rgba`. En esta notación el último argumento define el valor de la transparencia del color. Este último argumento tiene un valor comprendido entre 0.0 y 1.0.

```
contexto.strokeStyle = "rgba(195,215,235,0.5)"
```

Ejemplo



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#zone { border: 1px solid black; }
</style>
</head>
<body>
<canvas id="zone" width="200px" height="200px">
Su navegador no soporta la etiqueta canvas.
</canvas>
<script type="text/javascript">
var miCanvas = document.getElementById("zone");
var contexto = miCanvas.getContext("2d");
contexto.strokeStyle = "rgb(195,215,235)";
contexto.beginPath();
contexto.lineWidth = 20;
contexto.moveTo(50,25);
contexto.lineTo(50,175);
contexto.stroke();
contexto.beginPath();
contexto.moveTo(100,25);
contexto.lineWidth = 20;
contexto.lineTo(100,175);
contexto.globalAlpha = 0.5;
contexto.stroke();
contexto.beginPath();
contexto.strokeStyle = "rgba(195,215,235,0.5)";
contexto.moveTo(150,25);
contexto.lineWidth = 20;
contexto.lineTo(150,175);
contexto.stroke();
</script>
</body>
</html>
```

7. Diseñar formas múltiples

En todos los ejemplos anteriores siempre hemos diseñado formas, unas junto a otras. Pero es muy frecuente que las formas se superpongan. El API Canvas, con la propiedad `globalCompositeOperation`, permite determinar el comportamiento que deben adoptar estas formas superpuestas.

No solo es posible diseñar nuevas formas detrás de las formas existentes, sino también utilizarlas para ocultar algunas zonas, borrar algunas partes del lienzo y muchas más operaciones.

```
globalCompositeOperación = "tipo"
```

donde tipo es una cadena que representan una de las doce operaciones de composición que se describen más adelante.

En todos los ejemplos siguientes, el cuadrado se diseña en primer lugar y se llama al contenido existente del lienzo. Después se diseña el círculo y se llama a una nueva forma. Estos diseños se extraen de la documentación oficial de Firefox.

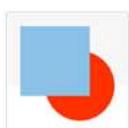
source-over

Es la configuración por defecto. Las nuevas formas se dibujan por encima del contenido existente en el lienzo.



destination-over

Las nuevas formas se dibujan detrás del contenido existente en el lienzo.



source-in

La nueva forma se dibuja solo cuando se superpone al lienzo de destino. El resto (aquí el cuadrado) se vuelve transparente.



destination-in

El contenido existente en el lienzo se conserva allá donde la nueva forma y el contenido existentes se superponen. El resto se vuelve transparente.



source-out

La nueva forma se dibuja allá donde no se superpone al contenido existente en el lienzo.



destination-out

El contenido existente se conserva allá donde no se superpone a la nueva forma.



source-atop

La nueva forma se dibuja únicamente allá donde se superpone al contenido existente en el lienzo.



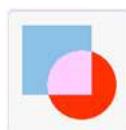
destination-atop

El lienzo existente solo se conserva allá donde se superpone a la nueva forma. Esta última se dibuja detrás del contenido del lienzo.



lighter

Allá donde las dos formas se superponen, el color resultante es la adición de los los valores de color.



darker

Allá donde las dos formas se superponen, el color resultante es la sustracción de los valores de color.



xor

Las formas se vuelven transparentes allá donde se superponen y se dibujan con normalidad en el resto.

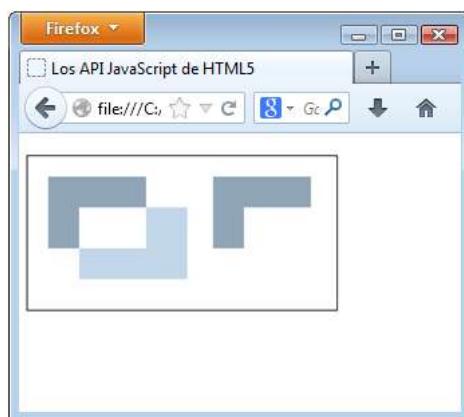


copy

Dibuja solo la nueva forma y borra el resto.



Ejemplo



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
```

```

<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#zone { border: 1px solid black; }
</style>
</head>
<body>
<canvas id="zone" width="300px" height="150px">
Su navegador no soporta la etiqueta canvas.
</canvas>
<script type="text/javascript">
var miCanvas = document.getElementById("zone");
var contexto = miCanvas.getContext("2d");
contexto.fillStyle="rgb(145,165,185)"
contexto.fillRect(20,20,95,70);
contexto.fillStyle="rgb(195,215,235)";
contexto.globalCompositeOperation="xor";
contexto.fillRect(50,50,105,70);
contexto.fillStyle="rgb(145,165,185)"
contexto.fillRect(180,20,95,70);
contexto.fillStyle="rgb(195,215,235)";
contexto.globalCompositeOperation="destination-out";
contexto.fillRect(210,50,105,70);
</script>
</body>
</html>

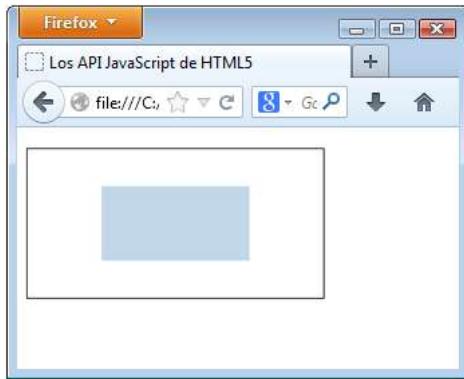
```

8. Colocación dinámica

Es posible ubicar un elemento de manera dinámica con JavaScript. Esto se hace con ayuda del método `translate(x, y)`.

Ejemplo

Vamos a centrar dinámicamente un rectángulo usando código JavaScript.



El código

```

<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#zone { border: 1px solid black; }
</style>
<script type="text/javascript">
window.onload = function() {
var lienzo = document.getElementById("zone");
var contexto = lienzo.getContext("2d");
var rectWidth = 150;
var rectHeight = 75;
contexto.translate(lienzo.width / 2,
lienzo.height / 2);
contexto.fillStyle = "rgb(195,215,235)";
contexto.fillRect(-rectWidth / 2, -rectHeight / 2, rectWidth,
rectHeight);
}
</script>
</head>
<body>
<canvas id="zone" width="300px" height="150px">
Su navegador no soporta la etiqueta canvas.
</canvas>
</body>
</html>

```

9. Registro y recuperación de argumentos

Los desarrolladores que deben gestionar diseños más complejos son aficionados a los siguientes métodos porque, cuando se asimilan correctamente, permiten ahorrar muchas líneas de código.

```

save()
restore()

```

Los métodos `save` (guardar) y `restore` (restaurar) del API Canvas se usan para registrar y recuperar el estado del lienzo. De manera resumida, un estado es una foto de todos los estilos y transformaciones que se han aplicado al lienzo. Estos dos métodos no tienen argumentos.

Los estados se almacenan en una pila. Cuando se llama al método `save`, el estado actual se añade a la pila. Un estado incluye:

- Las transformaciones que se han aplicado (es decir, las translaciones, rotaciones y escalado; ver en este capítulo las secciones Redimensionar una imagen y Rotar una imagen).

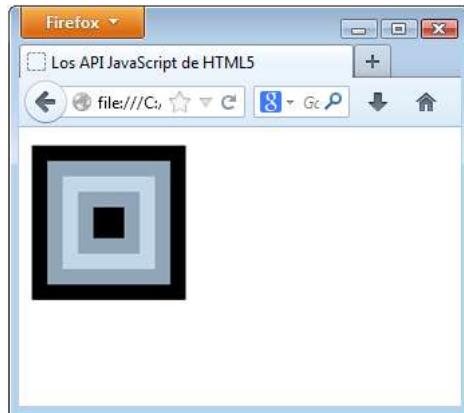
- Los valores de las propiedades `strokeStyle`, `fillStyle`, `globalAlpha`, `lineWidth`, `lineCap`, `lineJoin`, `miterLimit`, `shadowOffsetX`, `shadowOffsetY`, `shad`

El método `save` se puede usar tantas veces como se desee.

Cada vez que se llama al método `restore`, se elimina de la pila el último estado guardado y se restauran todos los argumentos.

Es importante entender que `save()` y `restore()` no guardan o restauran el contenido del lienzo en sí mismo. Estos métodos solo guardan y restauran las propiedades o atributos.

Ejemplo



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<script type="text/javascript">
function draw() {
var lienzo = document.getElementById("zone");
var contexto = lienzo.getContext("2d");
contexto.fillRect(0,0,150,150);
contexto.save();
contexto.fillStyle = "rgb(145,165,185)"
contexto.fillRect(15,15,120,120);
contexto.save();
contexto.fillStyle = "rgb(195,215,235)"
contexto.fillRect(30,30,90,90);
contexto.restore();
contexto.fillRect(45,45,60,60);
contexto.restore();
contexto.fillRect(60,60,30,30);
}
</script>
</head>
<body onload="draw()">
<canvas id="zone" width="200px" height="160px">
Su navegador no soporta la etiqueta canvas.
</canvas>
</body>
</html>
```

para determinar el color.

```
contexto.restore();
contexto.fillRect(60,60,30,30);
```

Y para terminar el cuadrado central, restauramos el primer estado, que era el de color negro. De esta manera, el cuadrado central es de color negro. Con esto nos ahorraremos otra línea de código.

Detallemos este código.

```
contexto.fillRect(0,0,150,150);
contexto.save();
```

El script diseña un rectángulo de 150 x 150. Se utiliza el color por defecto, es decir, el negro. Usando el método `save()`, se guarda este argumento.

```
contexto.fillStyle = "rgb(145,165,185)"
contexto.fillRect(15,15,120,120);
contexto.save();
```

Después se pinta un cuadrado más pequeño (120 x 120) con un color (`fillStyle`) azul medio. Este color se guarda con el método `save()`. Este estado se inserta en la pila.

```
contexto.fillStyle = "rgb(195,215,235)"
contexto.fillRect(30,30,90,90);
```

Pintamos un cuadrado todavía más pequeño (90 x 90) en un azul más claro.

```
contexto.restore();
contexto.fillRect(45,45,60,60);
```

El script restaura (`restore`) el estado anterior, es decir, el que está en primer lugar en la pila y que contiene el color azul medio. Se pinta un nuevo cuadrado (60 x 60) con este color. De esta manera, hemos ahorrado una línea de código

El texto

1. El texto sencillo

Para añadir texto en el lienzo, el diseñador dispone de los métodos `fillText("texto", x, y)` y `strokeText("texto", x, y)`, donde `texto` es el texto que se desea incluir, `x` la coordenada horizontal del inicio del texto e `y` la coordenada vertical del inicio del texto.

Ejemplo



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#zone { border: 1px solid black; }
</style>
</head>
<body>
<canvas id="zone" width="200px" height="120px">
Su navegador no soporta la etiqueta canvas.
</canvas>
<script type="text/javascript">
var miCanvas = document.getElementById("zone");
var contexto = miCanvas.getContext("2d");
contexto.fillText("El API Canvas de Html5",40,40);
contexto.strokeText("El API Canvas de Html5",40,80);
</script>
</body>
</html>
```

Pasemos rápidamente a los siguientes puntos para adornar este texto, quizás demasiado básico.

2. El tamaño y la fuente de caracteres

El tamaño y la fuente de caracteres se definen con la propiedad `font`.

La sintaxis es idéntica a la de la propiedad de estilo CSS font:

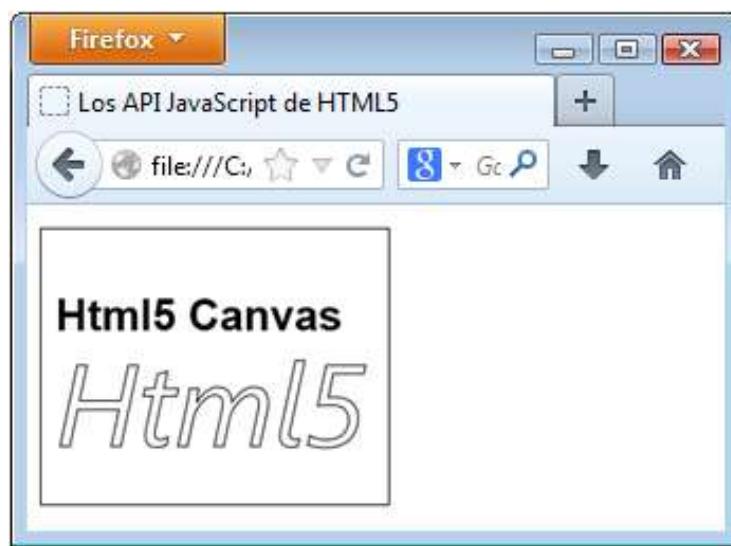
```
contexto.font = "Bold o Italic, tamaño, fuente"
```

Por ejemplo:

```
contexto.font = "20pt Arial";  
contexto.font = "Bold 20pt Arial";
```

El valor por defecto es 10px sans-serif.

Ejemplo



El código

```
<!DOCTYPE html>  
<html lang="es">  
<head>  
<title>Los API JavaScript de Html5</title>  
<meta charset=utf-8>  
<style type="text/css">  
#zone { border: 1px solid black; }  
</style>  
</head>  
<body>  
<canvas id="zone" width="240px" height="190px">  
Su navegador no soporta la etiqueta canvas.  
</canvas>  
<script type="text/javascript">  
var miCanvas = document.getElementById("zone");  
var contexto = miCanvas.getContext("2d");  
contexto.font = 'Bold 30px Sans-Serif';  
contexto.fillText("Html5 Canvas",10,70);  
contexto.font = 'Italic 80px "Segoe UI"';  
contexto.strokeText("Html5",10,150);  
</script>  
</body>  
</html>
```

► La diferencia entre `fillText` y `strokeText` aquí es más explícita. El método `fillText` muestra las letras totalmente rellenas, mientras que `strokeText` presenta únicamente el contorno de estas.

3. El color del texto

Para añadir color al texto del Canvas, es suficiente con utilizar la propiedad `fillStyle` que ya hemos explicado anteriormente.

Ejemplo



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#zone { border: 1px solid black; }
</style>
</head>
<body>
<br>
<canvas id="zone" width="350px" height="100px">
Su navegador no soporta la etiqueta canvas.
</canvas>
<script type="text/javascript">
var miCanvas = document.getElementById("zone");
var contexto = miCanvas.getContext("2d");
contexto.font= 'Bold 50px Sans-Serif';
contexto.fillStyle = "rgb(145,165,185)";
contexto.fillText ("Html5 Canvas",10,70);
</script>
</body>
</html>
```

4. Alinear el texto

La propiedad `textAlign` permite alinear el texto. Los valores posibles son:

- `left` para alinear a la izquierda.
- `right` para alinear a la derecha.
- `center` para centrar el texto.
- `start` (valor predeterminado) para alinear al inicio de la línea en la escritura de izquierda a derecha.
- `end` para alinear al final de la línea en la escritura de derecha a izquierda.

Por ejemplo: contexto.textAlign = "left";

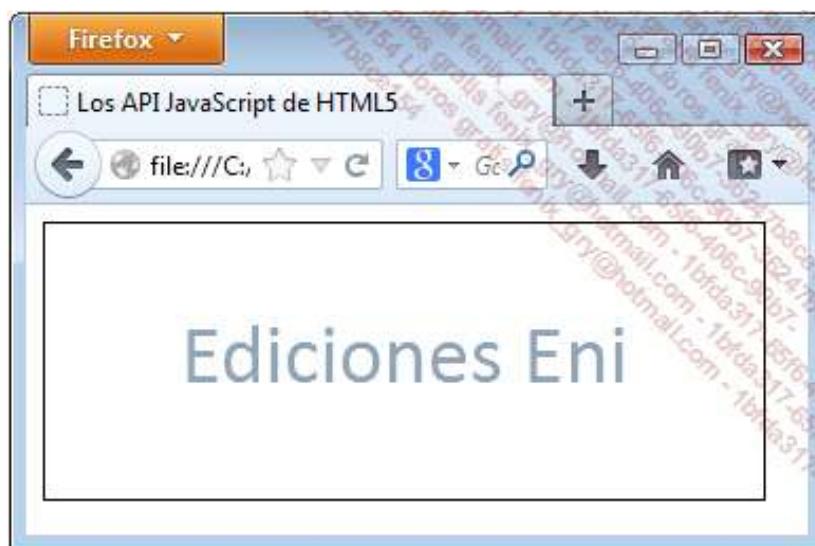
Para terminar, la propiedad textBaseline define la línea de referencia de la escritura del texto. Los valores posibles se ilustran en la siguiente imagen y son:

- top.
- middle.
- alphabetic (valor predeterminado).
- bottom.

Por ejemplo: contexto.textBaseline = "alphabetic";



Ejemplo



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#zone { border: 1px solid black; }
</style>
</head>
<body>
<canvas id="zone" width="340px" height="130px">
Su navegador no soporta la etiqueta canvas.
</canvas>
```

```

<script type="text/javascript">
var miCanvas = document.getElementById("zone");
var contexto = miCanvas.getContext("2d");
contexto.textBaseline = "alphabetic";
contexto.font = "30pt Calibri";
contexto.textAlign = "center";
contexto.fillStyle = "rgb(145,165,185)";
contexto.fillText("Ediciones Eni",170,75);
</script>
</body>
</html>

```

5. Sombreado

El sombreado se añade con las propiedades shadowOffsetX, shadowOffsetY, shadowBlur y shadowColor.

Las propiedades shadowOffsetX = valor y shadowOffsetY = valor indican el desplazamiento o offset del sombreado respecto a los ejes horizontal y vertical. Se pueden usar valores negativos para representar sombreados que se desplazan hacia arriba y a la izquierda. Los valores positivos proporcionan un efecto de sombreado hacia abajo y a la derecha. El valor por defecto es 0 para las dos propiedades.

La propiedad shadowBlur = valor determina el efecto de dispersión del sombreado. El valor por defecto es 0.

La propiedad shadowColor = "color" indica el color del efecto de sombreado. Este color se indica siguiendo la notación de las hojas de estilo CSS. Por defecto, el color es el negro (*black*).

Ejemplo



El código

```

<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#zone { border: 1px solid black; }
</style>
</head>

```

```

<body>
<canvas id="zone" width="240px" height="190px">
Su navegador no soporta la etiqueta canvas.
</canvas>
<script type="text/javascript">
var miCanvas = document.getElementById("zone");
var contexto = miCanvas.getContext("2d");
contexto.font= 'Bold 30px Sans-Serif';
contexto.shadowOffsetX = 5;
contexto.shadowOffsetY = 5;
contexto.shadowBlur = 5;
contexto.shadowColor = "#808080";
contexto.fillText("Html5 Canvas",10,70);
contexto.font = 'Italic 80px "Segoe UI"';
contexto.strokeText("Html5",10,150);
</script>
</body>
</html>

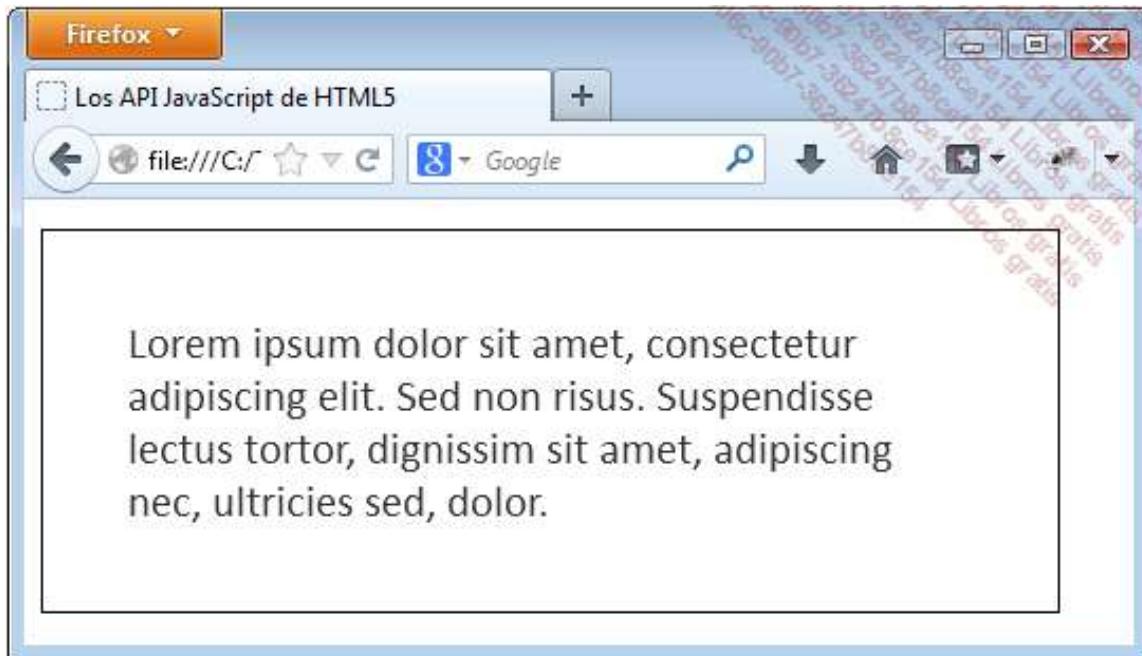
```

6. Añadir texto más largo

Para presentar de manera elegante texto más largo, podemos elaborar una función propia (`wrapTexto`), que tiene como argumentos el contexto del lienzo, el texto, una posición x e y, una anchura máxima y el interlineado. Esta función utilizará el método `measureText()` del API Canvas que calcula el retorno de carro o salto de línea.

```
wrapTexto(contexto, texto, x, y, anchura_máxima, interlineado);
```

Ejemplo



El código

```

<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#zone { border: 1px solid black; }
</style>
<script type="text/javascript">

```

```

function wrapTexto(contexto, text, x, y, maxWidth, lineHeight){
var words = text.split(" ");
var line = "";
for (var n = 0; n < words.length; n++) {
var testLine = line + words[n] + " ";
var metrics = contexto.measureText(testLine);
var testWidth = metrics.width;
if (testWidth > maxWidth) {
contexto.fillText(line, x, y);
line = words[n] + " ";
y += lineHeight;
}
else {
line = testLine;
}
}
contexto.fillText(line, x, y);
}

window.onload = function(){
var lienzo = document.getElementById("zone");
var contexto = lienzo.getContext("2d");
var maxWidth = 400;
var lineHeight = 25;
var x = (lienzo.width - maxWidth) / 2;
var y = 60;
var text = "Lorem ipsum dolor sit amet, consectetur adipiscing
elit. Sed non risus. Suspendisse lectus tortor, dignissim sit amet,
adipiscing nec, ultricies sed, dolor.";
contexto.font = "16pt Calibri";
contexto.fillStyle = "#333";
wrapTexto(contexto, text, x, y, maxWidth, lineHeight);
};

</script>
</head>
<body onmousedown="return false;">
<br>
<canvas id="zone" width="480" height="180">
Su navegador no soporta la etiqueta canvas.
</canvas>
</body>
</html>

```

7. Añadir efectos especiales

Hasta ahora hemos presentado ejemplos básicos. No podemos resistirnos a presentar algunos ejemplos infográficos creativos.



Puede ver otros ejemplos en la dirección del sitio Web [html5rocks](http://html5rocks.com/en/tutorials/canvas/texteffects/) (www.html5rocks.com/en/tutorials/canvas/texteffects/). Como guinda del pastel, le proponemos un archivo con los diferentes ejemplos para que pueda descargarlo. No deje de echar un vistazo al código fuente de estos ejemplos.

Las imágenes

1. Añadir una imagen

La importación de imágenes se realiza en dos fases:

- En primer lugar hay que crear una imagen como objeto JavaScript. No es posible incluir una imagen utilizando simplemente el atributo HTML `src`.
- La función `drawImage` se usa para diseñar la imagen en el lienzo.

Es importante verificar que la imagen se haya cargado correctamente antes de llamar a la función `drawImage`.

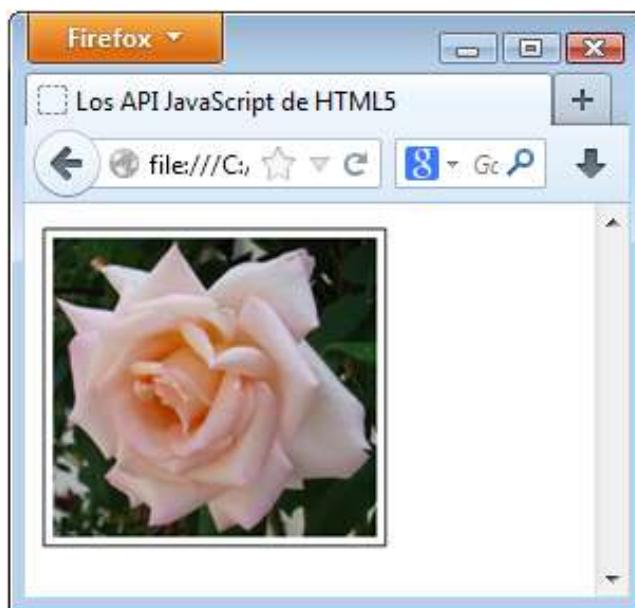
La sintaxis de esta función es:

```
drawImage(imagen, x, y)
```

donde `imagen` es una referencia a la imagen y `x` e `y`, las coordenadas de la ubicación de la imagen en el lienzo.

Las imágenes pueden tener formato GIF, JPEG o PNG.

Ejemplo



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#zone { border: 1px solid black; }
</style>
<script type="text/javascript">
window.onload = function(){
var lienzo = document.getElementById("zone");
var contexto = lienzo.getContext("2d");
var img = new Image();
img.onload = function() {
```

```

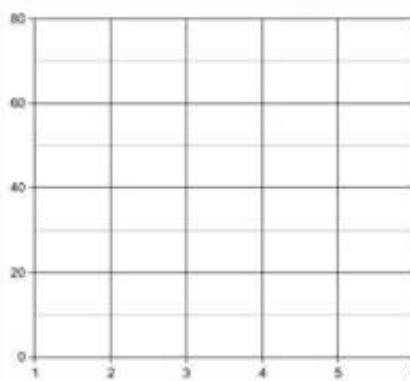
contexto.drawImage(img,5,5);
};

img.src = "rosa.png";
}
</script>
<body>
<canvas id="zone" width="210px" height="194px">
Su navegador no soporta la etiqueta canvas.
</canvas>
</body>
</html>

```

Por supuesto, es posible superponer líneas en una imagen.

Dada una imagen como fondo de gráfico:



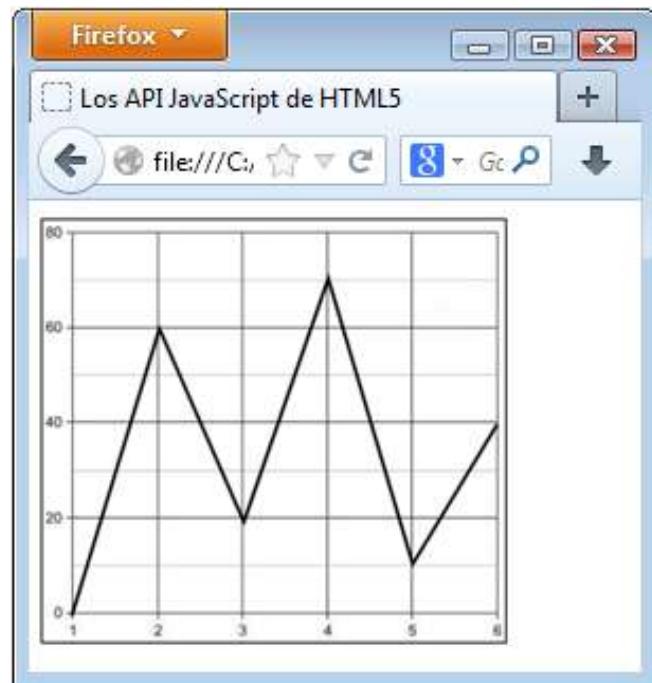
A esta imagen, incluida en el lienzo, le vamos a añadir una línea de gráfico.

```

<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#zone { border: 1px solid black; }
</style>
</head>
<script type="text/javascript">
function diseño() {
var lienzo = document.getElementById("zone");
var contexto = lienzo.getContext("2d");
var img = new Image();
img.src = 'grafico.png';
contexto.drawImage(img,0,0);
contexto.beginPath();
contexto.lineWidth = 2;
contexto.moveTo(18,244);
contexto.lineTo(72,67);
contexto.lineTo(124,186);
contexto.lineTo(176,36);
contexto.lineTo(228,212);
contexto.lineTo(280,126);
contexto.stroke();
}
</script>
<body onload="diseño()">
<canvas id="zone" width="285px" height="260px">
Su navegador no soporta la etiqueta canvas.
</canvas>


```

```
</body>  
</html>
```



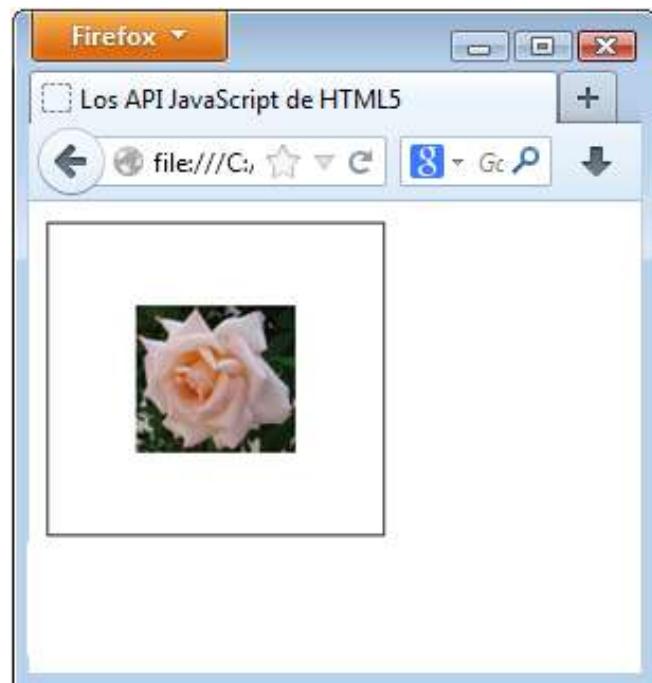
2. Redimensionar una imagen

Para modificar el tamaño de la imagen que se muestra en el lienzo, es suficiente con añadir dos argumentos en el método `drawImage()`.

La sintaxis se convierte en:

```
drawImage(imagen, x, y, anchura, altura)
```

Ejemplo



El código

```

<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#zone { border: 1px solid black; }
</style>
<script type="text/javascript">
window.onload = function(){
var lienzo = document.getElementById("zone");
var contexto = lienzo.getContext("2d");
var img = new Image();
img.onload = function(){
contexto.drawImage(img,55,51,100,92);
};
img.src = "rosa.png";
}
</script>
<body>
<canvas id="zone" width="210px" height="194px">
Su navegador no soporta la etiqueta canvas.
</canvas>
</body>
</html>

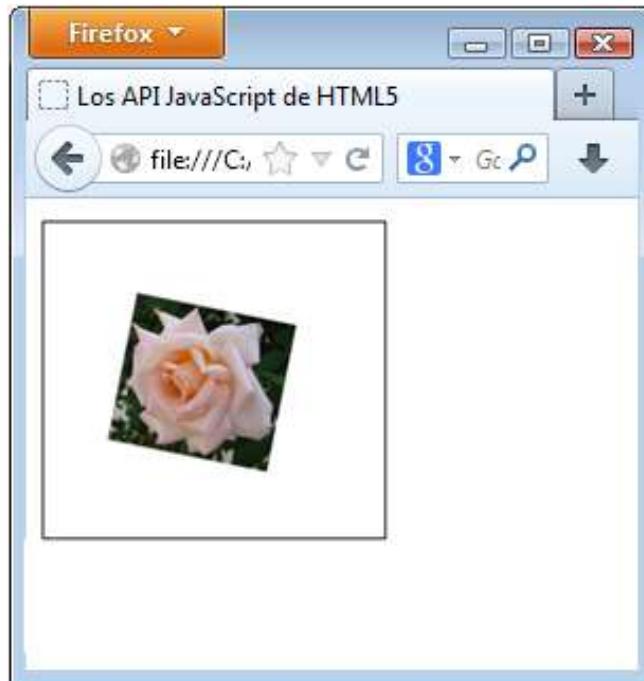
```

3. Rotar una imagen

Para realizar una rotación de una forma geométrica o de una imagen, dispone del método `rotate(angle)`, donde el ángulo se expresa en radianes.

```
contexto.rotate(0.05);
```

Ejemplo



El código

```

<!DOCTYPE html>
<html lang="es">
<head>

```

```

<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#zone { border: 1px solid black; }
</style>
<script type="text/javascript">
window.onload = function(){
var lienzo = document.getElementById("zone");
var contexto = lienzo.getContext("2d");
var img = new Image();
img.onload = function(){
contexto.drawImage(img, 65, 31, 100, 92);
};
img.src = "rosa.png";
contexto.rotate(0.20);
}
</script>
<body>
<br>
<canvas id="zone" width="210px" height="194px">
Su navegador no soporta la etiqueta canvas.
</canvas>
</body>
</html>

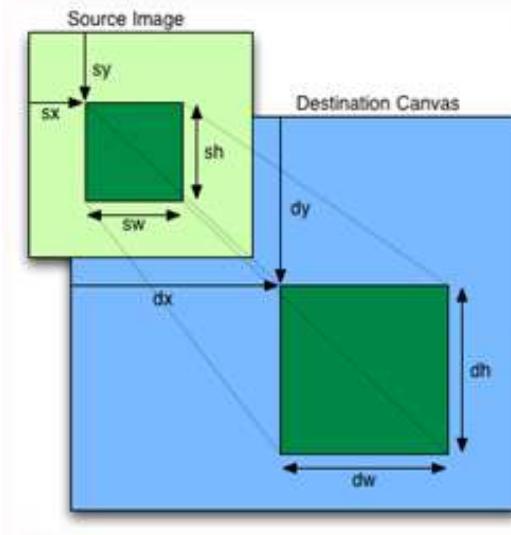
```

4. Recortar una imagen

Para conservar únicamente una parte de una imagen, es suficiente con añadir argumentos al método `drawImage()`: `sourceX`, `sourceY`, `anchura_origen`, `altura_origen`, `destinoX`, `destinoY`, `anchura_destino`, `altura_destino`. Estos argumentos definen la localización y el tamaño del rectángulo que queremos conservar de la imagen inicial.

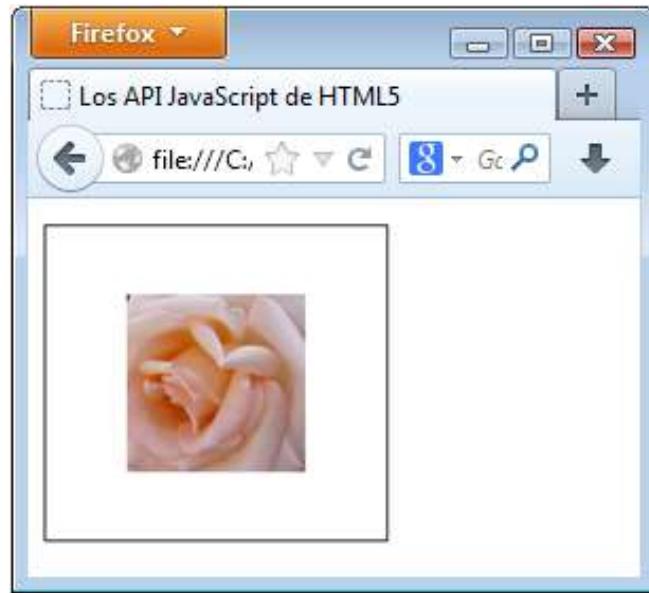
```
contexto.drawImage(image, sourceX, sourceY, anchura_origen,
altura_origen, destinoX, destinoY, anchura_destino, altura_destino);
```

El siguiente gráfico está extraído del sitio Web: www.html5canvastutorials.com e ilustra todo esto.



Ejemplo

De nuestra rosa, solo conservamos la parte central.



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#lienzo { border: 1px solid black; }
</style>
<script type="text/javascript">
window.onload = function(){
var lienzo = document.getElementById("lienzo");
var contexto = lienzo.getContext("2d");
var img = new Image();
img.onload = function(){
var sourceX = 40;
var sourceY = 35;
var sourceWidth = 110;
var sourceHeight = 110;
var destWidth = sourceWidth;
var destHeight = sourceHeight;
var destX = lienzo.width / 2 - destWidth / 2;
var destY = lienzo.height / 2 - destHeight / 2;
contexto.drawImage(img, sourceX, sourceY, sourceWidth,
sourceHeight, destX, destY, destWidth, destHeight);
};
img.src = "rosa.png";
};
</script>
<body>
<canvas id="lienzo" width="210px" height="194px">
Su navegador no soporta la etiqueta canvas.
</canvas>
</body>
</html>
```

5. Guardar un lienzo como una imagen

Es posible guardar un diseño Canvas como una imagen usando el método `toDataUrl()`. Sin embargo, estas imágenes deben provenir de un mismo dominio. El formato por defecto es PNG. El usuario puede guardar entonces la imagen en su ordenador haciendo clic en el botón derecho del ratón.

Consideremos el siguiente diseño realizado con el API Canvas:



Es posible guardar el lienzo como una imagen.



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#lienzo { border: 1px solid black; }
</style>
<script type="text/javascript">
window.onload = function(){
var lienzo = document.getElementById("lienzo");
var contexto = lienzo.getContext("2d");
contexto.beginPath();
contexto.moveTo(170, 80);
contexto.bezierCurveTo(130, 100, 130, 150, 230, 150);
contexto.bezierCurveTo(250, 180, 320, 180, 340, 150);
}
```

```
contexto.bezierCurveTo(420, 150, 420, 120, 390, 100);
contexto.bezierCurveTo(430, 40, 370, 30, 340, 50);
contexto.bezierCurveTo(320, 5, 250, 20, 250, 50);
contexto.bezierCurveTo(200, 5, 150, 20, 170, 80);
contexto.closePath() // complete custom shape
contexto.lineWidth = 5;
contexto.fillStyle = "rgb(195,215,235)";
contexto.fill();
contexto.strokeStyle = "rgb(165,185,205)";
contexto.stroke();
var dataURL = lienzo.toDataURL();
document.getElementById("canvasImg").src = dataURL;
};

</script>
</head>
<body onmousedown="return false;">
<canvas id="zone" width="500" height="200" style="display:none;">
Su navegador no soporta la etiqueta canvas.
</canvas>
<img id="canvasImg" alt="">
</body>
</html>
```

Comentario

```
var dataURL = lienzo.toDataURL();
```

La variable **dataURL** llama al método **toDataURL()** para el lienzo.

```
document.getElementById("canvasImg").src = dataURL;
```

A la imagen identificada por **canvasImg** se le asigna una dirección (**src**), de manera que se pueda guardar como una imagen.

Las animaciones

1. Introducción a la animación

Antes de empezar la codificación de las animaciones, se recomienda partir de un lienzo vacío. El método `clearRect()` permite eliminar todos los estados del lienzo.

La sintaxis es:

```
clearRect(0, 0, anchura_del_lienzo, altura_del_lienzo)
```

Ejemplo

```
contexto.clearRect(0, 0, 500, 500);
```

2. El API requestAnimationFrame para las animaciones

Con las animaciones, vamos más allá de los diseños estáticos para, como en un vídeo, visualizar un número de imágenes por segundo.

El API Canvas no se diseñó para las animaciones; las primeras de ellas resultaron decepcionantes y muy costosas en términos de recursos.

El API `requestAnimationFrame` se diseña para determinar el desplazamiento óptimo de las imágenes por segundo. En el estado actual del desarrollo del API, cada navegador hace su propia interpretación. Se usan los prefijos webkit para Google Chrome y Safari, moz para Firefox, o para Opera, y ms para Internet Explorer.

El marco de una animación contendrá el siguiente código:

```
window.requestAnimationFrame = (function(callback) {
  return window.requestAnimationFrame ||
  window.webkitRequestAnimationFrame ||
  window.mozRequestAnimationFrame ||
  window.oRequestAnimationFrame ||
  window.msRequestAnimationFrame ||
  function(callback) {
    window.setTimeout(callback, 1000 / 60);
  };
})();
```

Este código permite llamar al API en función del navegador.

```
function animate() {
  var canvas = document.getElementById("zone");
  var contexto = canvas.getContext("2d");
```

La definición clásica de un lienzo.

```
contexto.clearRect(0, 0, lienzo.width, lienzo.height);
```

Vaciado inicial de cada nuevo lienzo.

Después seguimos las instrucciones de diseño del lienzo.

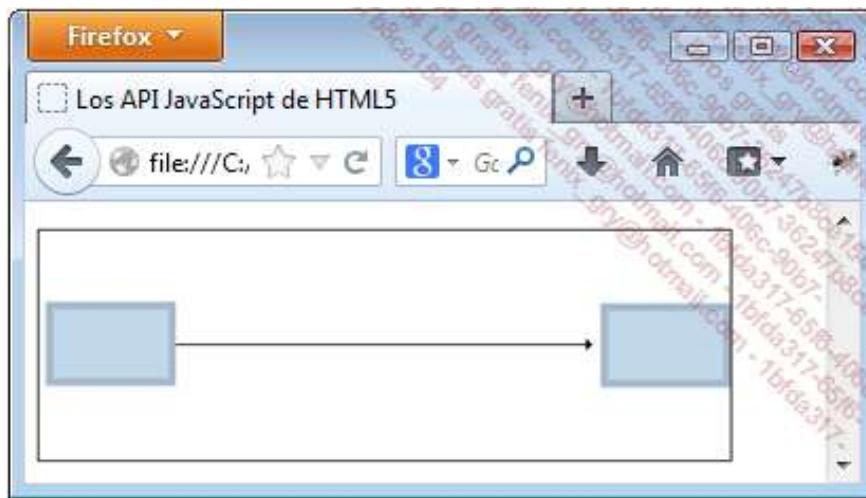
```
requestAnimationFrame(function() {
  animate();
});
```

Llamada del diseño o del siguiente lienzo.

3. Desplazamiento lineal

Ejemplo

Movemos un rectángulo de manera lineal.



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#lienzo { border: 1px solid black; }
</style>
<script type="text/javascript">
window.requestAnimFrame = (function(callback) {
return window.requestAnimationFrame ||
window.webkitRequestAnimationFrame ||
window.mozRequestAnimationFrame ||
window.oRequestAnimationFrame ||
window.msRequestAnimationFrame ||
function(callback) {
window.setTimeout(callback, 1000 / 60);
};
})();
function animate(lastTime, MiRectangulo) {
var lienzo = document.getElementById("lienzo");
var contexto = lienzo.getContext("2d");
var date = new Date();
var time = date.getTime();
var timeDiff = time - lastTime;
var linearSpeed = 100; // pixeles por segundo
var linearDistEachFrame = linearSpeed * timeDiff / 1000;
var currentX = MiRectangulo.x;
if (currentX < lienzo.width - MiRectangulo.width -
MiRectangulo.borderWidth / 2) {
var newX = currentX + linearDistEachFrame;
MiRectangulo.x = newX;
}
lastTime = time;
contexto.clearRect(0, 0, lienzo.width, lienzo.height);
}
</script>

```

```

contexto.beginPath();
contexto.rect(MiRectangulo.x, MiRectangulo.y, MiRectangulo.width, MiRectangulo.height);
contexto.fillStyle = "rgb(195,215,235)";
contexto.fill();
contexto.lineWidth = MiRectangulo.borderWidth;
contexto.strokeStyle = "rgb(165,185,205)";
contexto.stroke();
requestAnimationFrame(function() {
  animate(lastTime, MiRectangulo);
});
}
window.onload = function() {
var MiRectangulo = {
x: 0,
y: 50,
width: 80,
height: 50,
borderWidth: 5
};
var date = new Date();
var time = date.getTime();
animate(time, MiRectangulo);
};
</script>
</head>
<body onmousedown="return false;">
<br>
<canvas id="zone" width="450" height="150">
Su navegador no soporta la etiqueta canvas.
</canvas>
</body>
</html>

```

Comentario

```

window.requestAnimationFrame = (function(callback) {
  return window.requestAnimationFrame || ...
  function(callback) {
    window.setTimeout(callback, 1000 / 60);
  };
})();

```

Llamada del API requestAnimationFrame.

```

function animate(lastTime, MiRectangulo){
var lienzo = document.getElementById("zone");
var contexto = lienzo.getContext("2d");

```

Por cada llamada de la función animate (), se define un nuevo lienzo.

```

var date = new Date();
var time = date.getTime();
var timeDiff = time - lastTime;
var linearSpeed = 100; // pixeles por segundo
var linearDistEachFrame = linearSpeed * timeDiff / 1000;
var currentX = MiRectangulo.x;
if (currentX < lienzo.width - MiRectangulo.width -
MiRectangulo.borderWidth / 2) {
var newX = currentX + linearDistEachFrame;
MiRectangulo.x = newX;
}
lastTime = time;

```

Estas líneas de código gestionan el paso al siguiente lienzo en función de una cantidad de tiempo (`date.getTime()`) y determinan para este lienzo la nueva posición del rectángulo (`MiRectangulo.x`).

```
contexto.clearRect(0, 0, lienzo.width, lienzo.height);
```

Antes de realizar cualquier diseño en este nuevo lienzo, nos aseguramos de que está vacío (`clearRect`).

```
contexto.beginPath();
contexto.rect(MiRectangulo.x, MiRectangulo.y, MiRectangulo.width, MiRectangulo.height);
contexto.fillStyle = "rgb(195,215,235)";
contexto.fill();
contexto.lineWidth = MiRectangulo.borderWidth;
contexto.strokeStyle = "rgb(165,185,205)";
contexto.stroke();
```

Diseñamos el nuevo lienzo y, por tanto, un nuevo rectángulo. Observe la posición variable en el eje horizontal de este, mediante `MiRectangulo.x`.

```
requestAnimationFrame(function(){
animate(lastTime, MiRectangulo);
});
```

Ahora podemos reiniciar el proceso y devolver el control a la función `animate()`.

```
window.onload = function(){
var MiRectangulo = {
x: 0,
y: 50,
width: 80,
height: 50,
borderWidth: 5
};
var date = new Date();
var time = date.getTime();
animate(time, MiRectangulo);
};
```

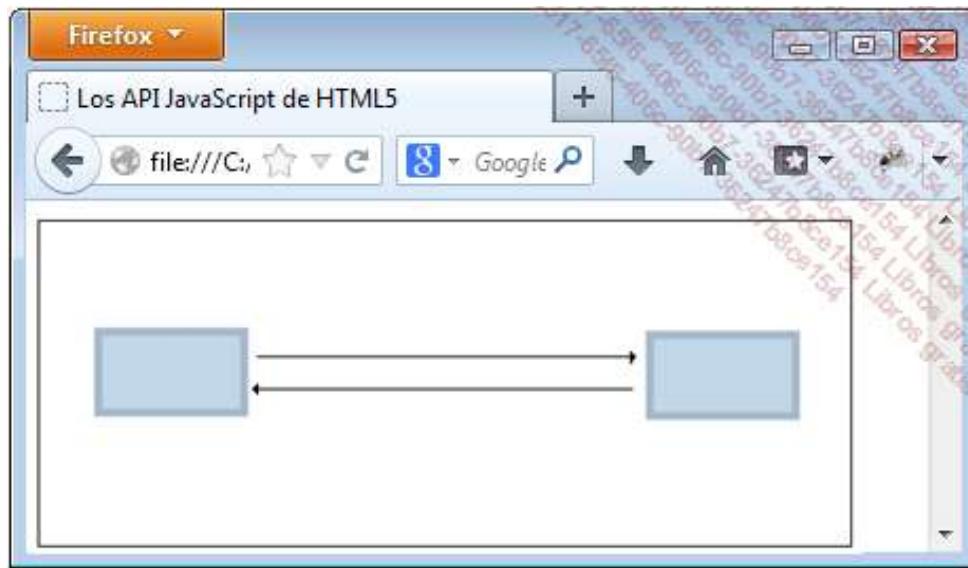
Al cargar la página, se definen diversas variables relativas al rectángulo y se fija el tiempo 0.

4. Desplazamiento lineal alternativo

Repetimos el mismo proceso, pero esta vez con un desplazamiento alternativo del rectángulo.

Para crear una animación con un efecto de oscilación, podemos utilizar la ecuación de un oscilador armónico sencillo para regular la posición de la forma para cada imagen, es decir, $x(t) = amplitud * \operatorname{sen}(t * 2\pi / periodo) + x_0$.

Ejemplo



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#zone { border: 1px solid black; }
</style>
<script type="text/javascript">
window.requestAnimFrame = (function(callback) {
return window.requestAnimationFrame ||
window.webkitRequestAnimationFrame ||
window.mozRequestAnimationFrame ||
window.oRequestAnimationFrame ||
window.msRequestAnimationFrame ||
function(callback) {
window.setTimeout(callback, 1000 / 60);
};
})();
function animate(MiRectangulo) {
var lienzo = document.getElementById("zone");
var contexto = lienzo.getContext("2d");
var date = new Date();
var time = date.getTime();
var amplitude = 180;
var periode = 2000; // en milisegundos
var centerX = lienzo.width / 2 - MiRectangulo.width / 2;
var nextX = amplitude * Math.sin(time * 2 * Math.PI / periode) + centerX;
MiRectangulo.x = nextX;
contexto.clearRect(0, 0, lienzo.width, lienzo.height);
contexto.beginPath();
contexto.rect(MiRectangulo.x, MiRectangulo.y, MiRectangulo.width,
MiRectangulo.height);
contexto.fillStyle = "rgb(195,215,235)";
contexto.fill();
contexto.lineWidth = MiRectangulo.borderWidth;
contexto.strokeStyle = "rgb(165,185,205)";
contexto.stroke();
requestAnimFrame(function() {
animate(MiRectangulo);
});
}
}
```

```
window.onload = function(){
var MiRectangulo = {
x: 250,
y: 70,
width: 90,
height: 50,
borderWidth: 5
};
animate(MiRectangulo);
};

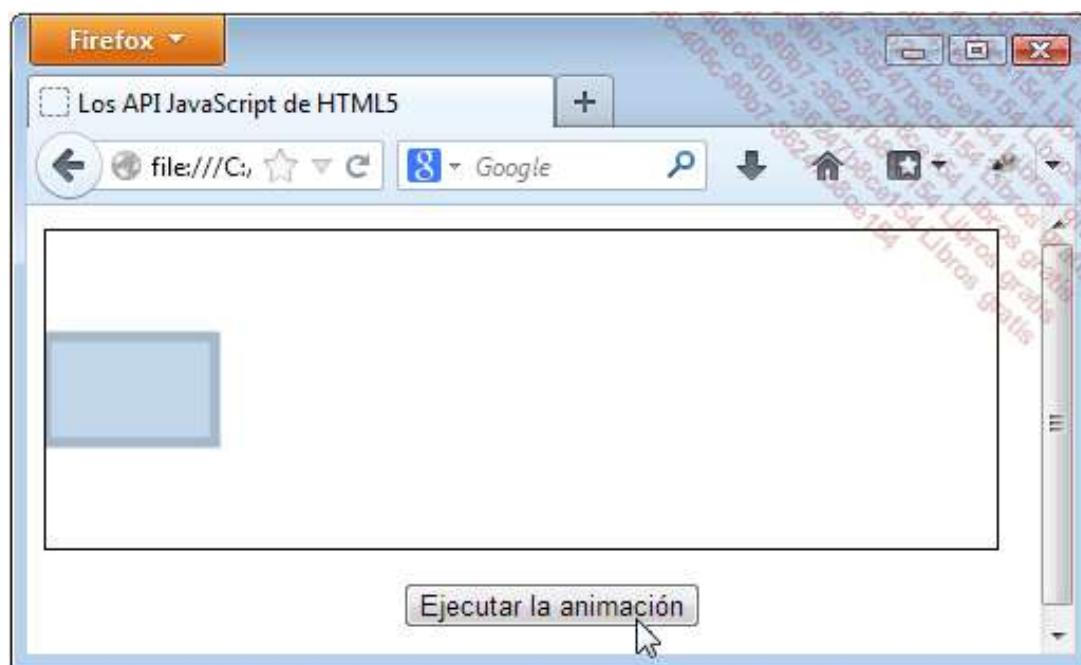
</script>
</head>
<body onmousedown="return false;">
<canvas id="zone" width="500" height="200">
Su navegador no soporta la etiqueta canvas.
</canvas>
</body>
</html>
```

5. Desencadenar una animación

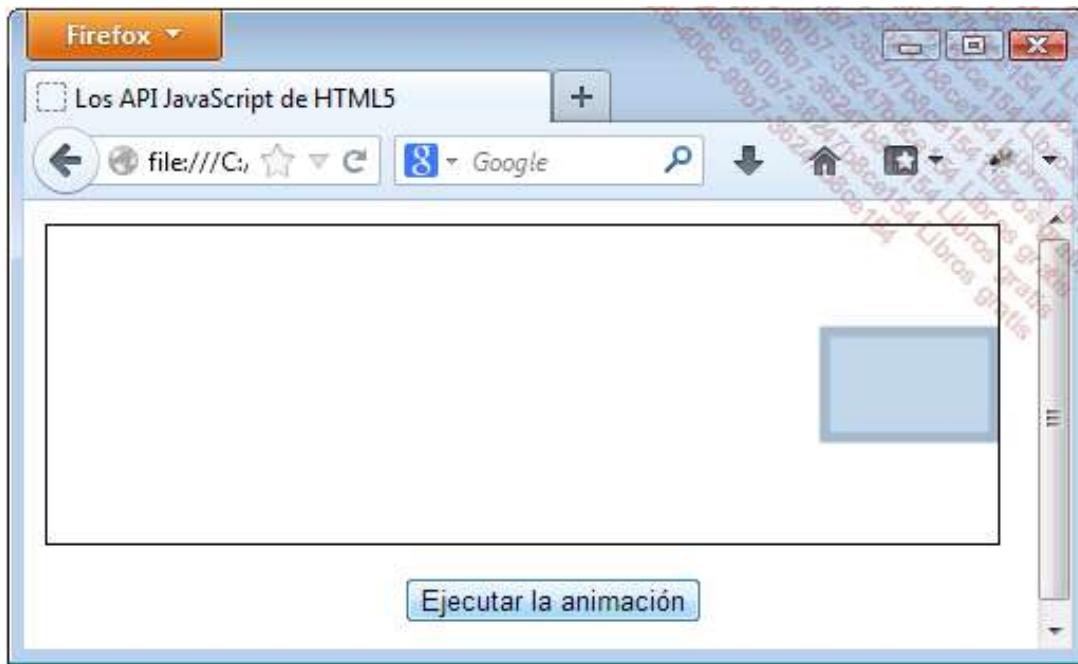
Y para terminar con las animaciones, quizás en lugar de ejecutarla cuando se carga la página, se puede dejar al usuario final elegir el momento en que desea desencadenarla, por ejemplo, haciendo clic en un botón.

Ejemplo

Inicio de la animación:



Al terminar:



El código

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Los API JavaScript de Html5</title>
<meta charset=utf-8>
<style type="text/css">
#lienzo { border: 1px solid black; }
#boton { margin-top: 10px;
margin-left: 170px; }
</style>
<script type="text/javascript">
window.requestAnimFrame = (function(callback) {
return window.requestAnimationFrame ||
window.webkitRequestAnimationFrame ||
window.mozRequestAnimationFrame ||
window.oRequestAnimationFrame ||
window.msRequestAnimationFrame ||
function(callback) {
window.setTimeout(callback, 1000 / 60);
};
})();
function drawRect(MiRectangulo){
var lienzo = document.getElementById("lienzo");
var contexto = lienzo.getContext("2d");
contexto.beginPath();
contexto.rect(MiRectangulo.x, MiRectangulo.y, MiRectangulo.width,
MiRectangulo.height);
contexto.fillStyle = "rgb(195,215,235)";
contexto.fill();
contexto.lineWidth = MiRectangulo.borderWidth;
contexto.strokeStyle = "rgb(165,185,205)";
contexto.stroke();
}
function animate(lastTime, MiRectangulo, animProp){
if (animProp.animate) {
var lienzo = document.getElementById("lienzo");
var contexto = lienzo.getContext("2d");
var date = new Date();
var time = date.getTime();
var timeDiff = time - lastTime;
```

```

var linearSpeed = 100; // pixeles por segundo
var linearDistEachFrame = linearSpeed * timeDiff / 1000;
var currentX = MiRectangulo.x;
if (currentX < lienzo.width - MiRectangulo.width -
MiRectangulo.borderWidth / 2) {
var newX = currentX + linearDistEachFrame;
MiRectangulo.x = newX;
}
lastTime = time;
contexto.clearRect(0, 0, lienzo.width, lienzo.height);
drawRect(MiRectangulo);
requestAnimationFrame(function() {
animate(lastTime, MiRectangulo, animProp);
});
}
}

window.onload = function(){
var MiRectangulo = {
x: 0,
y: 50,
width: 80,
height: 50,
borderWidth: 5
};
var animProp = {
animate: false
};
document.getElementById("button").addEventListener("click",
function(){
if (animProp.animate) {
animProp.animate = false;
}
else {
animProp.animate = true;
var date = new Date();
var time = date.getTime();
animate(time, MiRectangulo, animProp);
}
});
drawRect(MiRectangulo);
};
</script>
</head>
<body onmousedown="return false;">
<canvas id="zone" width="450" height="150">
Su navegador no soporta la etiqueta canvas.
</canvas><br>
<button id="boton">Ejecutar la animación</button>
</body>
</html>

```

Comentario

```

var animProp = {
animate: false
};

```

Gracias a las propiedades de animación, un manejador de eventos puede hacer referencia a un objeto.

```

document.getElementById("button").addEventListener("click",
function(){
...
});

```

Se añade un manejador de eventos de tal manera que, al hacer clic en el botón, se desencadena la animación.