La guía de la programación HTML5, CSS y JavaScript con Visual Studio

por Marino Posadas

Derechos reservados

Este libro se entrega en formato PDF para facilitarte el aprendizaje y que puedas copiar el código fuente de los ejemplos para tu uso personal.

Te pedimos que no hagas un uso ilegal de este libro, para poder seguir publicando libros tan especializados como este, para profesionales como tú.

Limitación de la responsabilidad

Tanto el autor como en Danysoft hemos revisado el texto para evitar cualquier tipo de error, pero no podemos prometerle que el libro esté siempre libre de errores. Por ello le rogamos nos remita por e-mail sus comentarios sobre el libro en attcliente@danysoft.com

Descuentos especiales

Danysoft ofrece descuentos especiales a centros de formación, a librerías, y en adquisiciones por volumen. Para más detalles, contacta con Danysoft.

Marcas registradas

Todos los productos y marcas se mencionan únicamente con fines de identificación y están registrados por sus respectivas compañías.

Datos del libro

Autor: Marino Posadas
Publicado por: Danysoft
Avda. de la Industria, 4 Edif. 1 | 28108 Alcobendas, Madrid. España.
902 123146 | www.danysoft.com
ISBN: | Depósito Legal: M-

A mis sobrinos, y -en especial- a Juan José Marcos García

A Milagros.

"Inside every well-written large program is a well-written small program."
- ***Charles Antony Richard Hoare***

"First, solve the problem. Then, write the code."
- ***John Johnson***

Introducción

Esta obra va dirigida a aquellos que quieren incorporar las nuevas capacidades que ofrece el estándar HTML5 y sus tecnologías vinculadas en aplicaciones y sitios Web modernos.

El único requisito es conocer los fundamentos de programación en los que se basa Internet (y –preferiblemente- algún lenguaje orientado a objetos). A partir de esas premisas, se recorre el estándar comenzando por el conjunto de etiquetas y atributos nuevos y modificados en HTML5, mostrando ejemplos de funcionamiento que el lector puede comprobar directamente sin más ayuda que un editor de texto y un navegador actualizado, aunque la obra también hace un recorrido por las herramientas, tanto de desarrollo y depuración, como **Visual Studio 2012/2013**, **Node.Js**, o **Fiddler**, como por las propias de los navegadores más populares.

A continuación se hace una revisión completa del lenguaje de presentación CSS3, analizando las partes clásicas que sirven de base a los desarrollos actuales, y comentando todas aquellas aportaciones que propone el estándar y ya pueden verse implementadas en los navegadores (o, al menos, en algunos de ellos). al igual que en el caso

anterior, cada escenario de presentación se ejemplifica para poder comprobar su funcionamiento actual.

Por último, se revisan los fundamentos del lenguaje JavaScript y –a continuación- las nuevas API que promueve el estándar, como siempre, ilustrando su funcionamiento con ejemplos de fácil implementación, que sirven como pruebas de concepto de cada una de las API que se estudian en este apartado.

La documentación básica en la que me he basado para el trabajo ha sido la publicada por la propia W3C, de forma que el estudio pudiera hacerse atendiendo al nivel de importancia de cada tecnología, independientemente de su implantación en un navegador concreto.

Agradecimientos

A Jose Luis Castaño de Danysoft por su confianza en mi trabajo y en esta nueva propuesta.

A los muchos amigos de Microsoft Ibérica que siempre me animan en estas labores de escritura y a los del grupo de MVP de Microsoft en España, con Cristina González Herrero a la cabeza.

A mis seguidores de Twitter, Facebook y otras redes sociales. Sus comentarios siempre son un acicate y una motivación para seguir.

Tabla de contenidos

Capítulo 01 | HTML 5: las tecnologías y su implicación en el desarrollo

Algunos divulgadores de las tecnologías que trataremos aquí, se refieren al estándar HTML5 como a una "marca", indicando que se trata de un apelativo global, utilizado para denominar un vasto conjunto de técnicas de desarrollo, encaminado a conseguir un objetivo: la "*Web Open Platform*" (Plataforma Web Abierta). Y entienden por tal, el enorme

agregado de especificaciones y recomendaciones que –una vez implementadas en las distintas plataformas y agentes de usuario- permitirán escribir código que pueda interpretarse de igual forma en cualquier dispositivo.

En otras palabras, el viejo sueño de programar una vez y ejecutar en cualquier sitio, que ya se propuso con la solución de las máquinas virtuales de ejecución o "*runtimes*", como el Java Runtime o .NET Framework. La diferencia aquí es que la propuesta implica considerar el navegador como una plataforma de ejecución, donde no sólo podemos ver páginas, sino ejecutar aplicaciones que se descargan desde algún servidor (o en la nube), y se ejecutan en el cliente (en el mejor de los casos, adaptándose al dispositivo de éste).

Para eso hay que llegar a un acuerdo entre los fabricantes de forma que todos los navegadores tengan –al menos- un patrón común que los desarrolladores puedan seguir. Y ese patrón es la promesa del estándar HTML5.

Como probablemente conocerá el lector, los estándares (los de Internet, al menos) son el resultado de un esfuerzo conjunto de entidades de todo tipo: las instituciones que se dedican a su confección y depuración, instituciones académicas, la industria, e incluso estudiosos particulares, aportan representantes que coordinan los esfuerzos comunes para llevar a buen puerto este barco abarrotado de documentos y propuestas. Estamos hablando de algo que –si lo viéramos impreso- abarcaría varios miles de folios escritos.

Así pues, no se trata de unas pocas novedades sino de un cambio muy profundo, y las últimas versiones de las herramientas de desarrollo más populares lo reflejan así.

El estado actual del estándar

No existe un único criterio rector o una sola influencia a la hora de abordar qué se incluye y qué se excluye de algo tan importante como el

comportamiento de la Web para los próximos años. De poco sirven las llamadas al consenso del propio **Tim Berners-Lee** (el inventor de la WWW), ya que siempre surgen disidencias debidas a intereses particulares o puntos de vista divergentes. Pero, disidencias aparte, al menos existe un canal principal de publicación, que nos permite consultar qué es lo que debiera estar presente y qué no.

El más importante de estos canales es, por supuesto, la propia W3C (*World Wide Web Consortium*), que lidera el citado Berners-Lee. Ellos se han encargado del proceso de estandarización de la Web desde los inicios y estándares como HTML 3.2, HTML 4.01, XML, XHTML, SVG, MathML y muchos otros son el resultado de su trabajo en los últimos 20 años[1].

Su Web la alberga el prestigioso *Massachussets Institute of Technology* a través de su Laboratorio de Informática (http://www.lcs.mit.edu/) en EE.UU., y dispone de réplicas en el INRIA francés (http://www.inria.fr), Universidad de Keio (http://www.keio.ac.jp/) en Japón y un conjunto de delegaciones por todo el mundo. Existen, además, otras organizaciones que coordinan actividades relacionadas de carácter abierto[2].

No obstante, no es la única entidad que propone y publica documentos de estandarización para Internet. Existe otra, la WHATWG[3] (*Web Hypertext Application Technology Working Group*) que es, según la Wikipedia: "una comunidad de personas interesadas en la evolución de HTML y las tecnologías conexas". Fue fundado por integrantes de Apple, la Fundación Mozilla y Opera Software.

Claro que, desde entonces, el editor encargado de publicar las propuestas de la WHATWG, **Ian Hickson**, ha pasado a trabajar en Google y declaraba[4] a comienzos de 2013 que su propósito es ahora distinto. En

---

[1] La especificación XML ya va por la 5ª edición y su documento oficial está disponible en http://www.w3.org/TR/2008/REC-xml-20081126/

[2] El sitio Web Identi.ca (http://identi.ca/w3ces) coordina las actividades de la W3C en España, que son –en su gran mayoría- de carácter abierto al público.

[3] http://www.whatwg.org/

[4] http://lists.w3.org/Archives/Public/public-whatwg-archive/2012Jul/0119.html

concreto busca: "promover un estándar sin número (solo HTML), de tipo dinámico, constantemente en proceso de cambio, según las necesidades o intereses lo aconsejen". Y añadía que "*no va a continuar con estos trabajos de coordinación*" con la W3C, a los que califica de "*venerables*"[5].

Hickson tampoco indica quién va establecer cuáles son esas necesidades o intereses que aconsejarán los cambios dinámicos en el estándar, aunque parece fácil de imaginar. Microsoft fue invitada a participar, pero declinó la oferta[6] al no contar con garantías por parte de WHATWG de que lo que se hiciese allí estaría libre de patentes, por lo que sigue colaborando oficialmente con la W3C.

No obstante, dado que la posición dominante de Google con el navegador podría suponer quedarse fuera del contexto, la empresa de Redmond anunció que "*soportaran cualquier aspecto de las nuevas propuestas, con tal de que haya –al menos- un par de navegadores que las soporten*" (independientemente de que formen parte o no del estándar). Debido a ello, en la reciente versión del navegador IE (v.11.0), algunas propuestas promovidas por Google, como **OpenGL**, están soportadas siendo la solución de compromiso, al menos de momento.

Especificaciones y pruebas de laboratorio

Microsoft se adhirió a esta filosofía cuando creó el eslogan "*Same Markup*", una idea del equipo de IE pergeñada durante el desarrollo de IE9, que pretende ser una propuesta a favor de la simplificación del desarrollo web con un objetivo: que todo el código HTML, CSS, JavaScript, etc., sea interpretado de forma idéntica independientemente

[5] Sorprende el uso de este término con carácter despectivo, dado que, tanto el diccionario Collins como el de la Real Academia Española de la Lengua, definen el término como "Digno de veneración, de respeto".

[6] http://blogs.msdn.com/b/cwilso/archive/2007/01/10/you-me-and-the-w3c-aka-reinventing-html.aspx

del navegador que la muestre. Por otro lado, la forma en que se van modelando las especificaciones a lo largo del tiempo puede dividirse en dos partes: la teórica y la práctica.

Desde el lado teórico, como hemos apuntado, es la colaboración y las aportaciones de sus miembros directos, que sugieren las ideas y también revisan las contribuciones y sugerencias de los participantes de carácter abierto, que son varios cientos en todo el mundo[7]. No existen votaciones, sino consenso a la hora de aprobar o desestimar un aspecto cualquiera del estándar. De esta forma, si un solo participante sugiere algo tecnológicamente válido y apropiado al marco del estándar, tiene muchos visos de ser aceptado, aunque haya partido originalmente de una sola fuente.

En total, parecen ser más de 100 las especificaciones agrupadas bajo el concepto (la marca) HTML5 y podrían incrementarse en el futuro antes de llegar al estado "*Last Call*". Lo abultado de este número se debe a que, bajo este nombre, tienen cabida cosas relacionadas pero muy distintas, como el código de marcado en sí (HTML), las hojas de estilo en cascada (CSS), el mecanismo de representación gráfica vectorial (SVG), los motores de interpretación de JavaScript, las API implementadas en este lenguaje y ya aprobadas (o en fase aprobación), etc. Esto plantea un serio reto respecto a las pruebas de funcionamiento, que son llevadas a cabo por las entidades participantes y publicadas oportunamente en los sitios Web de referencia.

[7] En España, destaca el grupo de trabajo vinculado a la Universidad Politécnica de Madrid, con una docena de miembros, bajo la dirección de la responsable de la Web Semántica en esta institución, Carmen Costilla.

El problema de los formatos

Hablar de HTML5 es hablar también de un protocolo de transporte de información que soporta ciertos formatos de datos (HTTP). Y aquí aparecen nuevos problemas, ya que los últimos movimientos en este sentido no ofrecen demasiada tranquilidad.

Me explico: el estándar oficial de archivos multimedia MP3/MP4 (técnicamente llamado H.264/MPEG-4 AVC), ha sido desarrollado conjuntamente por el ITU-T *Video Coding Experts Group* (VCEG) y el ISO/IEC *Moving Picture Experts Group* (MPEG), y es bien conocido y utilizado a nivel mundial. Algo similar ocurre con los formatos gráficos, donde encontramos estándares "*de facto*", como JPG y otros que ya son estándares "*de iure*" (oficiales), como el PNG (*Portable Network Graphics*), promovido por la misma W3C, como formato idóneo para Internet.

Sin embargo, en su búsqueda de mejoras para la rapidez de descargas de contenido en la Web, Google ha hecho esfuerzos por mejorar los formatos de audio y vídeo (formatos Vorbis y VP8/VP9, englobados en la denominación **WebM**), que, si bien son de código abierto y ahorran espacio de almacenamiento, no son estándares. Hasta aquí, esto no sería un problema. El problema aparecía con el anuncio de que las intenciones de la compañía son dejar el soporte de H.264, y soportar únicamente sus propios códec. Y algo parecido sucede con los formatos de imágenes. Recientemente, en la *Keynote* del evento "Google I/O 2013", se anunciaba la implantación de un nuevo formato de gráficos (**WebP**), que optimiza las imágenes en más de un 25% de media sin pérdida aparente de calidad. ¿El problema? Que pueden funcionar únicamente en Chrome si los demás navegadores no le dan soporte. Esperemos que exista acuerdo final entre todos los protagonistas y estos formatos se vean soportados por todas las plataformas.

Y todavía hay más: el motor *Webkit* utilizado por Chrome en la interpretación de las páginas Web hasta el momento, ha sido abandonado en favor de una nueva versión: *Blink*. El navegador Opera lo utilizaba igualmente, así como otros (Maxthon, Midori), y algunas

versiones de iOS. Parece que uno de los portavoces de Opera se ha apresurado a indicar que ellos también cambiarán al nuevo modelo, abandonando *Presto* (el que usaban hasta ahora), y colaborando con *Blink*, aunque alguno se pregunta por las repercusiones de este cambio[8].

En el capítulo siguiente veremos un cuadro de la situación actual de la implementación y recomendaremos algunos sitios Web dedicados al seguimiento del estándar en todas sus modalidades y su nivel de implantación en los navegadores (tanto el actual como el esperado). Tengamos en cuenta que también hay incompatibilidades por parte de otros navegadores, en especial en lo tocante al soporte de aspectos colaterales o más puntuales.

HTTP 2.0: Un nuevo protocolo de transporte

Hacemos aquí un inciso de algo que nos parece esclarecedor respecto a los numerosos movimientos en busca de mejoras en las experiencias de usuario en la red. Y es que, recientemente, se ha presentado la versión 2.0[9] del estándar de transporte HTTP, cuyo objetivo es conseguir que la carga de las páginas en los navegadores sea más rápida y eficiente, sin que eso suponga una pérdida en las características de seguridad.

Esta propuesta, basada en una iniciativa anterior de Google (la llamada SPDY[10], diseñada específicamente para transporte de contenido Web), está siendo implementada por el grupo *Hypertext Transfer Protocol Bis* (*HTTPBis*), que no es sino un grupo de trabajo de la entidad *Internet*

[8] Will Chrome for iOS be able to use Google's new browser rendering engine B*link?*: http://thenextweb.com/apple/2013/04/04/will-chrome-for-ios-be-able-to-use-googles-new-browser-rendering-engine-blink/

[9] http://tools.ietf.org/html/draft-ietf-httpbis-http2-04

[10] http://en.wikipedia.org/wiki/SPDY

Engineering Task Force, que coopera activamente con la W3C y las entidades de normalización ISO/IEC.

Los objetivos de este nuevo protocolo incluyen conexión asíncrona multiplexada, compresión de cabeceras, "*pipelining*" de tipo petición/respuesta y múltiples mensajes concurrentes sobre la misma conexión, lo que implica reducción de los períodos de latencia y tiempos de carga. También introduce la posibilidad de envíos tipo "*push*" (el servidor envía a los clientes, y no al revés) a clientes cualesquiera.

Dado el soporte inicial que el estándar ha encontrado por parte de Google, Microsoft, Firefox y Opera, es de esperar que las próximas actualizaciones importantes de los navegadores dispongan de este tipo de funcionalidad, y posiblemente de algunas API vinculadas con ella, que permitan la configuración y programación de estas posibilidades.

Técnicamente hablando, no se trata de parte del estándar, que solo se preocupa de los "lenguajes" de la red, pero, en su objetivo final, está esa mejora de la experiencia de usuario y del tiempo de respuesta, que redunda en la manera en que consumimos la web desde cualquier dispositivo.

El grado de madurez y finalización del estándar

A esta situación hay que añadir otra, propia de la naturaleza de lo que los fabricantes están creando: el nivel de implantación no es uniforme. Los navegadores incorporan en sus actualizaciones aspectos del estándar según su grado particular de avance en la incorporación de las novedades, sin que esto suceda al mismo tiempo.

Esta situación, resulta especialmente notable en 2 de los 3 protagonistas del estándar: CSS3 y las API de JavaScript. Especialmente en el primero, el nivel de implantación depende de muchos factores, por no mencionar que el propio estándar está subdividido en pequeños fragmentos con distintos grados de finalización.

Nivel de compleción de HTML5

En este sentido, la parte más madura y de la que no se esperan más cambios que los ya publicados, es HTML5. Con esto, nos referimos al lenguaje de marcas (nuevas etiquetas), incluyendo sus atributos y los valores que éstos pueden tener. A esto hay que añadir que la gran mayoría de las etiquetas ya existentes (versión HTML 4.01), ahora admiten algunos de esos atributos y valores. Con esta solución, se aporta valor a las páginas con marcado antiguo que simplemente quieran actualizar su código sin hacer cambios significativos.

El propósito anunciado por la W3C es que el HTML5 esté totalmente completado para 2014, si bien, salvo un par de inclusiones que tuvieron lugar recientemente (la etiqueta ***<main>*** y el atributo ***translate***), se da prácticamente por completado, a pesar de que su nivel oficial de finalización se encuentre en el estado "*Working Draft*" (Borrador de Trabajo). De hecho, la etiqueta ***<main>*** no ha sido aún aceptada oficialmente, si bien forma parte de la siguiente especificación, HTML 5.1, de la que ya disponemos de un borrador muy reciente[11].

El lenguaje de marcas fue lo primero que los navegadores se apresuraron a implementar y se encuentra disponible en más del 95% (más o menos…) en todos los navegadores modernos: IE9/IE10/IE11, Chrome, Safari, FireFox, Opera 11+, Konqueror, Maxthon, etc.

Las partes que cada fabricante tiene aún que implementar se encuentran relacionadas con implementaciones visuales de algunas etiquetas (aunque eso es opcional al fabricante, con tal de que la funcionalidad que aporta la etiqueta esté presente), así como algunas API relacionadas con grupos de etiquetas que aportan funcionalidad extra, como sucede con las relativas a las validaciones.

No obstante, se considera que la parte del marcado está operativa a todos los aspectos prácticos, especialmente, si tenemos en cuenta que existen *fallbacks* (explicamos este concepto más adelante), que permiten

[11] http://www.w3.org/html/wg/drafts/html/master/

aportar funcionalidad equivalente en versiones mucho más antiguas de los navegadores y otros agentes de usuario.

Nivel de compleción de CSS3

Estos grados de finalización o "estado de madurez" de las propuestas, son catalogados por la entidad mediante 5 criterios con la siguiente denominación (que vemos en la figura 1):

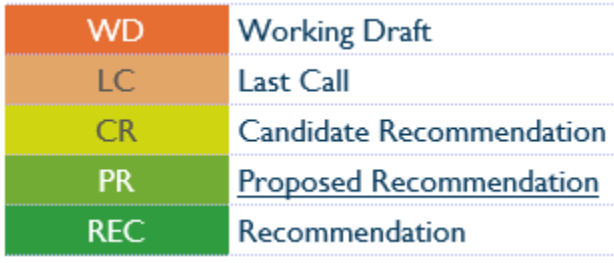


Fig. 1: Denominaciones usadas por la W3C para indicar el grado de madurez de una propuesta

Se corresponden con "Borrador de Trabajo", "Última Llamada", "Recomendación Candidata", "Recomendación Propuesta" y "Recomendación", siendo ésta última la que corresponde al estado final que debe ser implementado por los "Agentes de Usuario" (recordemos que los navegadores son solamente uno de los posibles mecanismos con los que se puede navegar por la Web).

En principio, sólo la última da garantías de implantación sin cambios. En la práctica, desde que se llega al estado "*Last Call*", se puede comenzar la implantación, dado que son mínimos los cambios que cabe esperar desde esa situación de Última Llamada, hasta el de Recomendación.

Un vistazo a la página oficial denominada del "estado actual" (*Current Work*[12]) del estándar, nos da pistas sobre esta situación:

[12] http://www.w3.org/Style/CSS/current-work

Completed	Cur-rent	Upcom-ing	Notes	
CSS Snapshot 2010	NOTE	–	Latest stable CSS	ⓘ
CSS Snapshot 2007	NOTE	–		ⓘ
CSS Color Level 3	REC	REC	See Errata	ⓘ
CSS Namespaces	REC	REC		ⓘ
Selectors Level 3	REC	REC		ⓘ
CSS Level 2 Revision 1	REC	REC	See Errata	ⓘ
CSS Level 1	REC	–	Unmaintained, see Snapshot	ⓘ
CSS Print Profile	NOTE	–		ⓘ
Media Queries	REC	REC		ⓘ

Fig. 2: Elementos del estándar CSS3 en estado "completado" (completed).

El gráfico anterior muestra aquellas recomendaciones ya concluidas por parte de la W3C. Eso significa que –al menos, en teoría- lo indicado en el gráfico de la figura 2 debiera estar presente en todos los navegadores modernos, y de hecho, salvo algún mínimo detalle, así es.

Y ¿qué pasa con el resto? Pues que los creadores de esos "agentes de usuario" van implementándolo a medida de sus necesidades y según los visos de viabilidad que la propuesta tenga. Hay que tener en cuenta que dentro de este maremágnum de propuestas, hay algunas que no llegan nunca a implementarse, bien porque son subsumidas por otras más completas, bien por otros problemas como puede ser la aparición de "agujeros" de seguridad.

En la realidad, los fabricantes han implementado ya la mayoría de las que aparecen en la figura 2 (pertenecientes a los estados CR, PR y LC).

El resto, que podemos ver simplificado en la figura 3 (junto a otros que no incluimos aquí), irá siendo implementado mediante este protocolo de "implementación y mejora continua" que es el nuevo "mantra" que se repite sin parar en los mensajes de las grandes compañías de software.

Por otro lado, tenga en cuenta el lector que se han establecido jerarquías de trabajo, estableciendo cuáles eran los aspectos del estándar que –de

cara a la implementación final- necesitaban de una solución más urgente.

Stable	Current	Upcoming
CSS Style Attributes	CR	PR
Testing	**Current**	**Upcoming**
CSS Backgrounds and Borders Level 3	CR	LC
CSS Conditional Rules Level 3	CR	CR
CSS Image Values and Replaced Content Level 3	CR	PR
CSS Marquee	CR	PR
CSS Multi-column Layout	CR	CR
CSS Speech	CR	PR
CSS Values and Units Level 3	CR	PR
CSS Flexible Box Layout	CR	PR
CSS Mobile Profile 2.0	CR	PR
CSS TV Profile 1.0	CR	?

Fig. 3: Elementos del estándar CSS3 en estado CR, PR o LC.

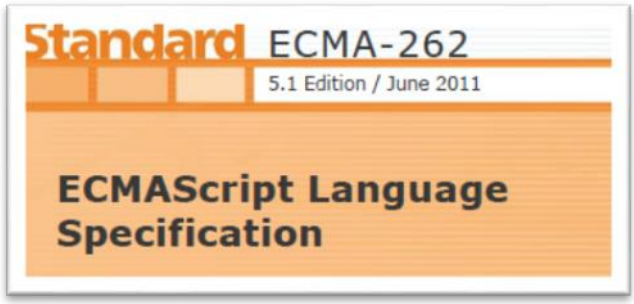
Refining	Current	Upcoming
CSS Animations	WD	WD
CSS Fonts Level 3	LC	CR
CSS Counter Styles Level 3	LC	CR
CSS Text Level 3	WD	WD
CSS Fragmentation Level 3	WD	WD
CSS Text Decoration Module Level 3	LC	CR
CSS Transforms	WD	WD
CSS Transitions	WD	WD
Cascading Variables	WD	LC
CSS Writing Modes Level 3	WD	WD

Fig. 4: Algunos elementos del estándar CSS3 se encuentran en estado "Working Draft", catalogados como "en proceso de refinamiento (refining).

Como hemos indicado, hay muchos otros, que la W3C cataloga en estadios anteriores a este, como "*Revising*" (Revisando), "*Exploring*" (Investigando posibilidades), y "*Rewriting*" (Rediseñándolo). El documento concluye con una pequeña lista de aquellos que han sido abandonados ("*Abandoned*") por razones como las que indicamos más arriba.

Nivel de compleción en JavaScript

Para el estándar JavaScript la situación es bastante mejor. Por una parte, la entidad de normalización encargada de la parte funcional del estándar (ECMA) llevaba bastante tiempo trabajando en la nueva versión de



JavaScript[13], y el grupo de trabajo que coordina las labores de sincronización con la W3C (llamado *TC39* o *Ecma Technical Committee 39*), ha estado trabajando en las API relacionadas con el nuevo estándar desde hace varios años (lo que implica docenas de miles de test sobre esas mismas API).

El comité de miembros permanentes está compuesto por representaciones de los principales líderes de la industria, y el grupo publica periódicamente sus avances en los sitios Web de la organización. Los anexos D y E de dicho documento incluyen posibles aspectos de esta versión del lenguaje que podrían tener un impacto de cara a la compatibilidad con versiones anteriores.

Las API de JavaScript

HTML y sus tecnologías no contribuirían mucho al objetivo de construir aplicaciones si no fuera por la presencia de API que aportan lo que ningún lenguaje de marcas o definiciones puede aportar. Por eso, conviene distinguir las API del lenguaje en sí, aunque la forma de programarlas sea mediante JavaScript.

Las API definen objetos que pueden usarse en conjunción con elementos del DOM (*Modelo de Objetos de Documento*) para completar su propuesta funcional o crear mecanismos complementarios. De ahí, se derivan posibilidades como la gestión del estado de las aplicaciones, la geo-localización, las aplicaciones *Offline*, las comunicaciones dúplex, el almacenamiento local y de sesión, el acceso a una base de datos sencilla,

[13] El documento oficial, ya concluido está disponible "on-line" en http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf

el soporte de tareas asincrónicas, etc.

En ocasiones, también se encuentran con problemas, o propuestas que deben de ser abandonadas a favor de otras que gozan de mejor implantación, seguridad, aceptación por la comunidad de colaboradores, etc.

Eso ha sucedido con la propuesta *WebSQL*, por ejemplo, que fue abandonada a favor de **IndexedDB**, o con la propuesta de **Web Sockets** que tuvo que ser revisada en su totalidad por problemas de seguridad (ya resueltos). Ésta última, una de las más atractivas, ha sido ya objeto de atención de muchas compañías que están implementando librerías y soluciones basadas en ella.

Una API de éxito: Web Sockets

En el caso de Microsoft, la propuesta se denomina **SignalR** y es una librería que permite la implementación de forma sencilla de esta API de forma integrada con cualquier aplicación utilizando directamente Visual Studio en cualquier versión a partir de la 2010. Además, esta librería se integra perfectamente con el IDE y permite incorporar esta API en cualquier solución ASP.NET nueva o existente, de forma sencilla, aportando opciones complementarias como el control de errores.

Otras API de interés están siendo igualmente adoptadas, en especial, aquellas que permiten dar un paso adelante en la solución de viejos problemas como el del almacenamiento de la información del usuario. A este respecto, tanto ***LocalStorage*** como ***SessionStorage*** suponen un paso adelante en este sentido y –además- su utilización es sumamente sencilla.

También disponemos ahora de nuevas propuestas en relación con la utilización de aplicaciones "Off-line", la Geo-localización, las características de "Arrastrar y Soltar" (tanto desde una misma página, como desde el sistema de archivos del usuario local). Todas ellas se encuentran ya implantadas en todos los navegadores modernos y para todas ellas existen alternativas de soporte cuando necesitemos incluirlas en navegadores antiguos.

La Experiencia de Usuario

Otro de los aspectos de implantación avanzada que se han visto reforzados (y en los que el estándar hace mucho hincapié en todos sus documentos finales) es el de la experiencia de usuario. Debe ser fluida, no debe bloquear jamás la interfaz de usuario, y debe ser adaptable al contexto (al dispositivo que alberga el mecanismo de navegación). Para ello, la nueva API de ***Web Workers*** y la versión de ***AJAX Level 2*** (*XHR2*), también suponen un paso adelante.

En resumen, prácticamente todas las API se encuentran implantadas en los agentes de usuario actuales, y –en todos los casos- encontramos alternativas en forma de librerías de terceros "*open source*" que nos ofrecen el soporte necesario en caso de que nuestra aplicación tenga que dar cobertura a navegadores antiguos (la mayor parte de ellas, por supuesto, basadas en soluciones JavaScript).

Hemos citado aquí algunas de las más importantes, aunque no todas. Las principales propuestas ya están definidas, pero el conjunto de API de JavaScript se considera un trabajo en constante desarrollo, y a medida que los borradores de cada especificación vayan pasando al estado *Recommendation*, veremos nuevas implantaciones.

La prueba de que esto es así la hemos podido ver recientemente en la última versión del navegador de Microsoft (IE11), con el soporte del estándar ***WebGL*** para la creación de gráficos de alto rendimiento. Y hay muchas más que se encuentran en estado "propuesta", por lo que cabe

esperar que, a partir de ahora, la evolución no sea un documento cerrado, sino, más bien, un proceso de "evolución continua", especialmente en este apartado de las librerías funcionales.

Las Herramientas

En el apartado de herramientas, la proliferación de sitios Web basados en los nuevos estándares ha supuesto igualmente un crecimiento exponencial de posibilidades y opciones disponibles.

Por ejemplo, si ya la versión 2012 de Visual Studio contaba con un excelente soporte del estándar, la versión 2013 *Preview* que utilizaremos a lo largo de esta obra da un paso más en todos los sentidos.

Visual Studio 2013 y el estándar HTML5

Aunque a la hora de escribir estas líneas la versión 2013 de Visual Studio no está disponible en su versión final, la *Preview* que Microsoft presentó en el evento *Tech-Ed* de Madrid, nos ofrece un anticipo bastante fiel de lo que será una vez concluida, y, especialmente, de las características que aporta al desarrollador Web, junto al soporte del estándar HTML5 y sus tecnologías.

Visual Studio es una herramienta muy extensible, que permite la adición de complementos desarrollados por terceras partes (o incluso por los usuarios). El acceso a estas extensiones lo tenemos disponible dentro de la opción "*Extensiones y Actualizaciones*" dentro del menú *Herramientas*.

Una vez aquí, disponemos de una gran cantidad de complementos que permiten la edición de código o el soporte de algunas opciones que no aparecen de forma nativa dentro del entorno, como comprimir imágenes, o las opciones de CSS que nos permiten saber qué versiones de los navegadores soportan una característica del estándar, como nos ofrece el complemento Web Essentials creado por **Mad Christensen**, del equipo de desarrollo de ASP.NET, que él mantiene actualizada para las distintas versiones de Visual Studio. En la imagen siguiente podemos este

complemento actualizable de apenas 1Mb de tamaño, tal y como se presenta en Visual Studio 2013.

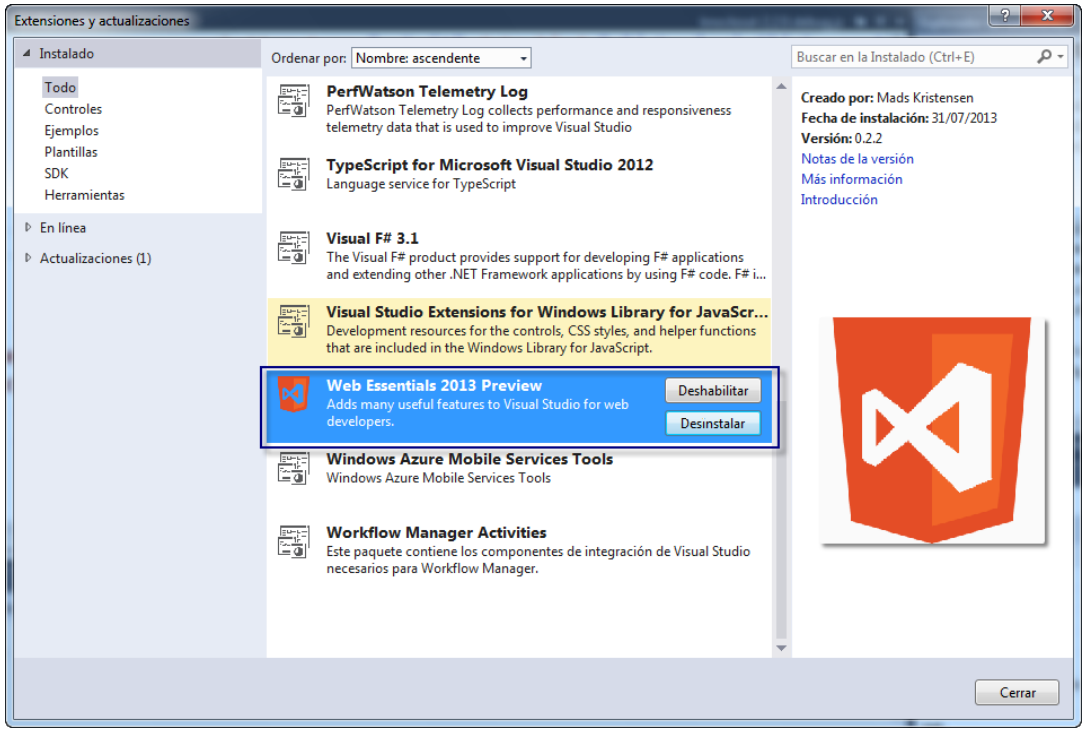


Fig. 5: El complemento Web Essentials desde la opción "Extensiones y Actualizaciones" de Visual Studio 2013.

Soporte del lenguaje HTML5

El soporte del estándar está garantizado en todas sus versiones con solo seleccionar la versión del lenguaje que deseamos utilizar en el menú "*Esquema de Destino para la Validación*", tal y como podemos ver en la figura 6.

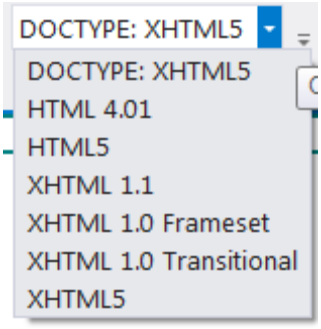


Fig. 6. Opción de "Esquema de destino en la validación", en Visual Studiio 2012/13, mostrando las opciones de esquema disponibles relacionadas con el estandar a soportar por la página en edición.

Este soporte incluye reconocimiento de las etiquetas propias de la especificación, incluyendo las partes relativas a los atributos (variables en muchos casos). También disponemos de soporte sobre los valores vinculados a cada atributo, cuando estos son de tipo booleano o conjuntos enumerados, como se ve en la figura siguiente.

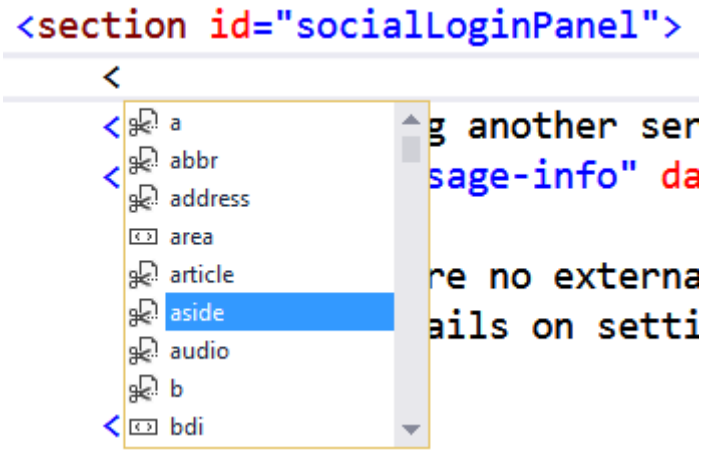


Fig. 7: Soporte de las etiquetas estándar de HTML 5 en Visual Studio (2010/12/13)

Es más, el propio IDE lo utiliza en plantillas nuevas de ASP.NET, como es el caso de las aplicaciones SPA (*Single Page Applications*), que son la recomendación más utilizada en casi todos los documentos oficiales de

Microsoft en este momento.

Y, como hemos apuntado antes, el soporte de los atributos de estas etiquetas es igualmente completo, como muestra la siguiente figura (fig. 8), mostrando el conjunto de atributos que dan soporte al marcado semántico:

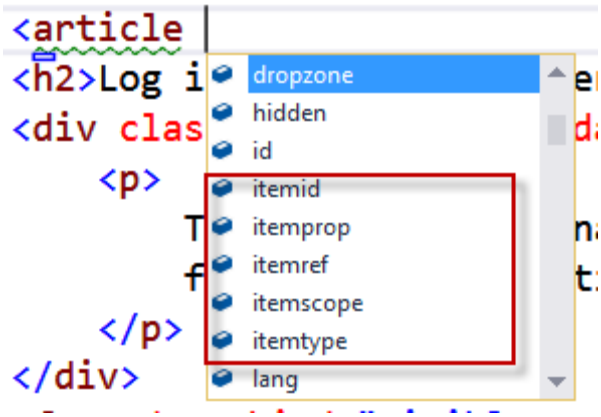


Fig. 8. Soporte de los nuevos atributos definidos por el estándar.

Otra opción muy solicitada por los usuarios era la inclusión de aspectos de refactorización dentro del código HTML. Eso nos permite marcar un bloque de elementos HTML y optar por cualquiera de las opciones de refactorización, como "*Envolver con*" o "*Insertar Snippet*" o elegir cualquiera de las posibilidades que nos ofrece *Refactoring*. Por ejemplo, si queremos incluir un bloque de párrafos (elementos **<p>**) dentro de un elemento ** **, podemos marcar todos los párrafos y seleccionar "*Envolver con…*" ("*Surround With*"), seguido de HTML y la etiqueta que sirve de contenedor, como vemos en la figura 9:

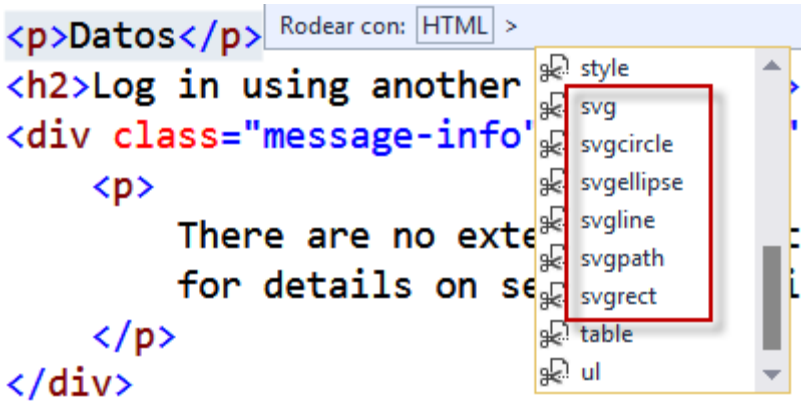


Fig. 9: La opción "Rodear con HTML" mostrando el soporte del estándar HTML5

Además, esta *Preview* ofrece varios anticipos interesantes, como es el soporte de **Angular.js**, una de las librerías de servidor más populares en la actualidad, que permite una integración sencilla con contextos de Bases de Datos y similares, y habilita su uso mediante etiquetas que tienen un prefijo específico (**ng-***), como podemos ver en la figura 10, cuando editamos una etiqueta HTML en el editor de Visual Studio 2013:

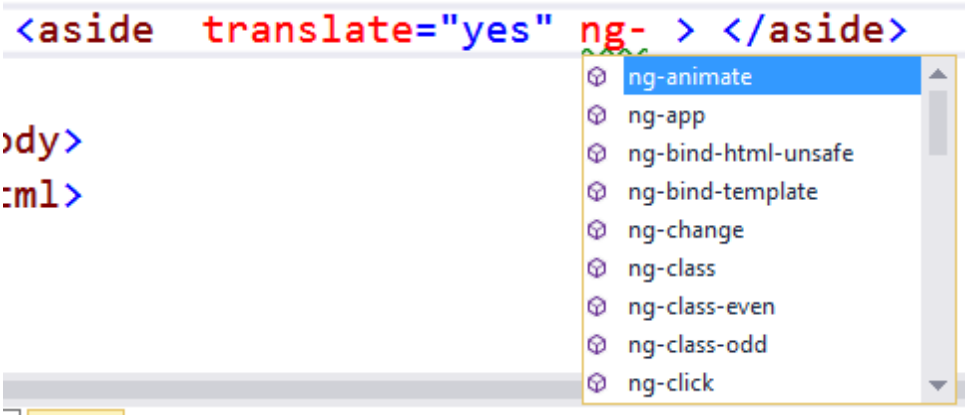


*Fig. 10: Editor de visual Studio 2013 mostrando el soporte de Angular.js basado en los prefijos de etiqueta que comienzan con ng-**

La figura anterior, además del soporte de esta librería, muestra algunas de las nuevas propuestas recientemente incorporadas al estándar, como el atributo **translate**, que establece si deseamos o no que los servicios

de traducción de páginas web se activen automáticamente al abrir una página.

Y lo mismo podemos decir de otras librerías, como es el caso de **Knockout**, presente al crear aplicaciones Web basadas en el modelo *Single Page Applications*, que estamos usando en esta demostración, tal y como muestra el Explorador de Soluciones de Visual Studio (Figura 11):

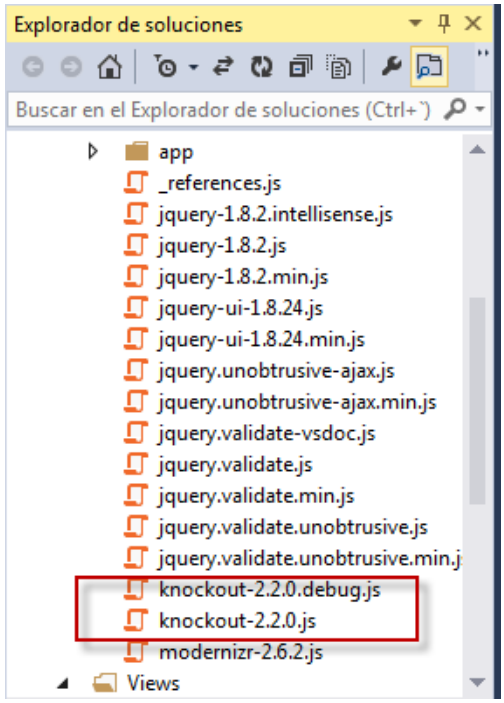


Fig. 11: Uso de las librerías JQuery, Knockout y Modernizr en Visual Studio 2013 Preview.

El Inspector de Páginas (Page Inspector)

En las versiones de Visual Studio 2012/2013, disponemos también de una herramienta muy útil para la depuración, integrada dentro del IDE: el **Inspector de Páginas** (*Page Inspector*). Es una herramienta de análisis dinámico que nos permite comprobar la ubicación, maquetación, estilos y comportamiento de cada elemento de marcado desde una ventana empotrada en Visual Studio.

Además, dispone de ventanas que habilitan la selección del tipo de inspección, dispone de un buscador de correspondencias CSS, de una herramienta de seguimiento de estilos (en la ventana "*Trace Styles*"), y de la posibilidad de ver cómo se mostraría la página en otro de los navegadores que tengamos instalados. Podemos verlo en funcionamiento en la figura 12.

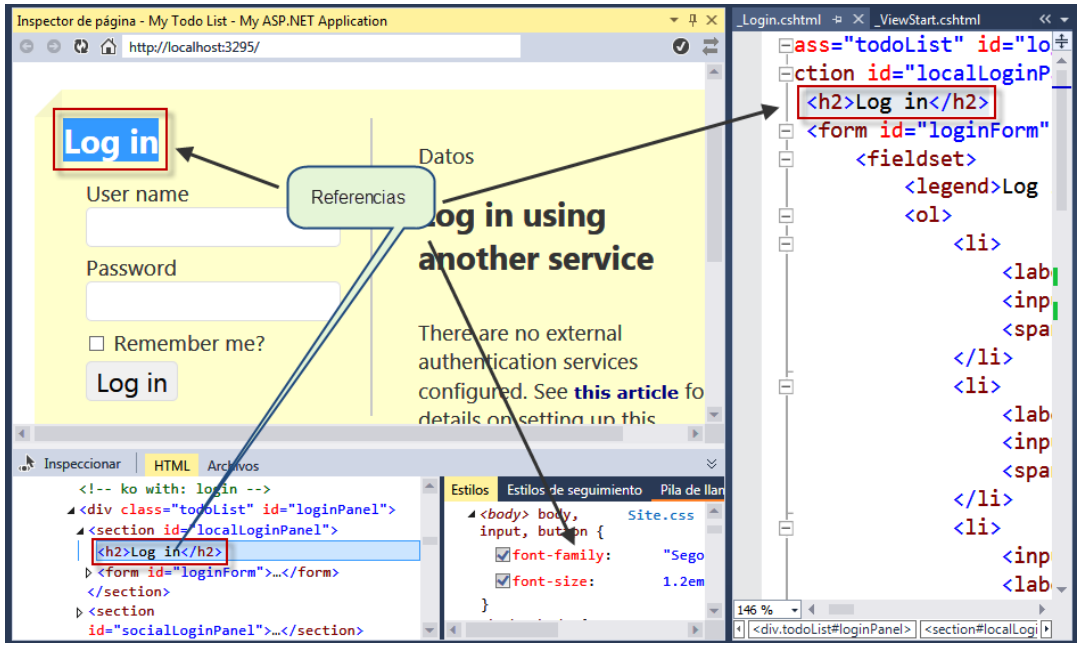


Fig. 12: El Inspector de Página, mostrando las referencias a un elemento.

En realidad, desde la versión V. Studio 2012, podemos ejecutar la aplicación Web directamente en *Page Inspector*, en lugar de seleccionar un navegador concreto, lo que nos permite ver simultáneamente el código fuente generado y el código original (y esto es válido igualmente para ASP.NET en cualquiera de sus variantes).

De forma que si inspeccionamos cualquier elemento, nos mostrará el elemento en el código generado, sus valores en ese instante y, simultáneamente, tenemos la posibilidad de localizarlo en el código original con que se corresponde. Una opción que no estaba presente en los depuradores anteriores a Visual Studio 2012.

Las ventanas de edición de sintaxis CSS también permiten cancelar o modificar la aplicación de estilos y ver el resultado en plena ejecución. Se pueden modificar atributos o eliminarlos temporalmente (en cualquier elemento) y ver el resultado en la ventana de ejecución. Con esta solución, disponemos de una alternativa muy completa a las herramientas de los navegadores, pero desde el propio Visual Studio.

Tendremos ocasión de revisar los detalles de este soporte al trabajar con estilos en los capítulos siguientes.

Lo mismo se puede decir la inspección de los ficheros que componen una vista cualquiera. Disponemos, además, de la posibilidad de ver todas las vistas relacionadas, a través de panel "Archivos", como aparece en la figura adjunta:

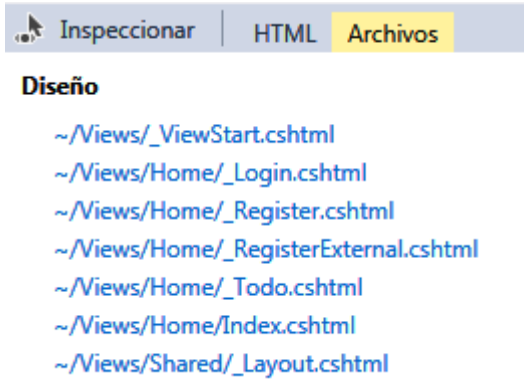


Fig. 13: El panel "Archivos" dentro del Inspector de Páginas

El soporte de JavaScript también se encuentra potenciado con la vista "*Pila de Llamadas*" que nos indica los ficheros cargados y las funciones que contienen, como nos muestra la figura 14.

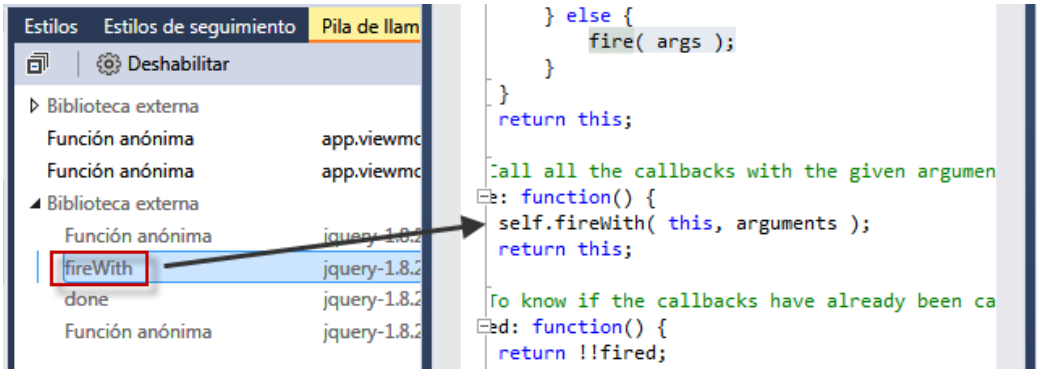


Fig. 14. Soporte de JavaScript en la ventana "Pila de Llamadas" del Inspector de Páginas

Vamos a ver a continuación el manejo del lenguaje JavaScript desde Visual Studio, ya que es el lenguaje que más novedades aporta en las últimas versiones del IDE.

El soporte de JavaScript en Visual Studio

Aparte de lo visto en el Inspector de Páginas, JavaScript es –de pleno derecho- un ciudadano de primera clase dentro del ecosistema de lenguajes de Visual Studio (aunque, naturalmente sea un lenguaje interpretado y no tenga nada que ver con .NET). Contamos con un soporte completo desde el punto de vista de la edición (*IntelliSense*, compleción de código, depuración, compresión de archivos para la distribución final con las opciones de "*bundling* & *minifying*", etc.).

De hecho, el motor de JavaScript se construyó para la versión IE9 completamente desde cero (implementación *Chakra*) y ofrece una rapidez sin comparación con la de cualquier otro motor del mercado (y ha seguido mejorando en las versiones siguientes IE10 e IE11).

Además, el mecanismo extendido de edición de código (*Intellisense*) reconoce los caracteres introducidos por el usuario, ofreciendo en el menú contextual solamente aquellas palabras que contienen el texto introducido hasta el momento, dependiendo de dónde nos encontremos (es sensible al contexto). También se han incorporado las características

de verificación de código ("*syntax checking*") que se encuentran presentes en otros lenguajes. En la figura 15 podemos ver dos de estas novedades: por un lado, el soporte de la función **trim**() definida en la especificación del nuevo ECMAScript 5, y por otro, la lista reducida de valores posibles que ofrece el editor:

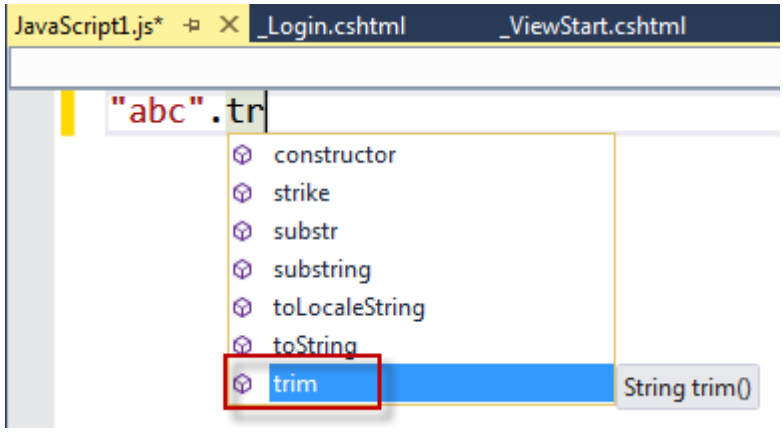


Fig. 15: El Editor de JavaScript mostrando opciones avanzadas

El mecanismo de *Intellisense*, además, reconoce las funciones definidas en otros archivos distintos del que nos encontramos y ofrece soporte sintáctico como si fuera un fragmento de código local. Además, se ha incluido en el menú contextual del editor la opción "*Ir a Definición*", de forma que podemos desplazarnos fácilmente a cualquier definición de otra función JavaScript, dondequiera que esté ubicada. Podemos ver una muestra de reconocimiento de la signatura de funciones en la figura 16:

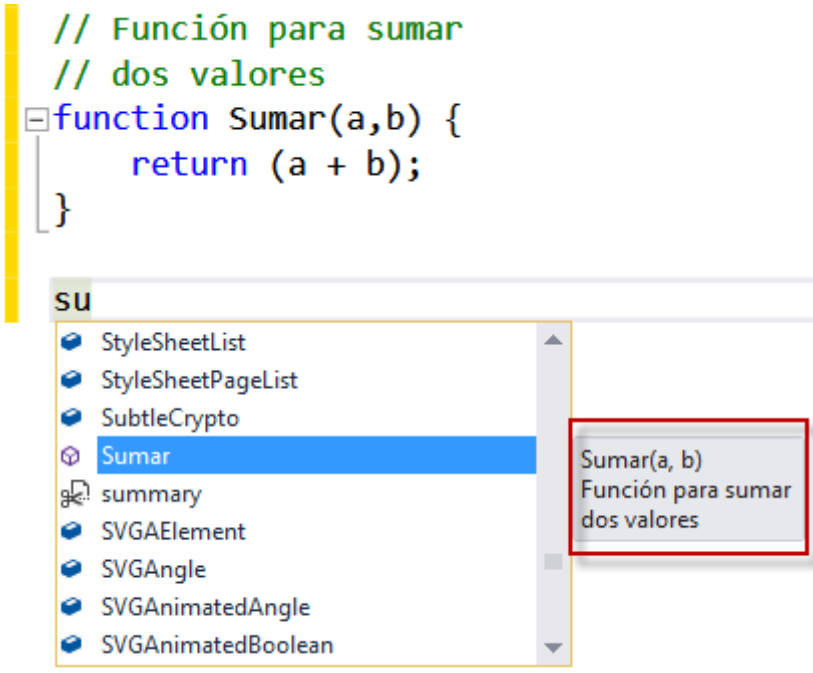


Fig. 16: Intellisense asociado a JavaScript

La opción de localizar la definición de una llamada en JavaScript es especialmente importante cuando queremos identificar operaciones almacenadas en librerías de terceros, como es el caso de jQuery.

En relación con esta última opción, disponemos de la posibilidad de indicar al editor que mantenga referencias implícitas sobre librerías. Ya contiene varias referencias predeterminadas, y podemos incluir las que necesitemos, como muestra la figura 17:

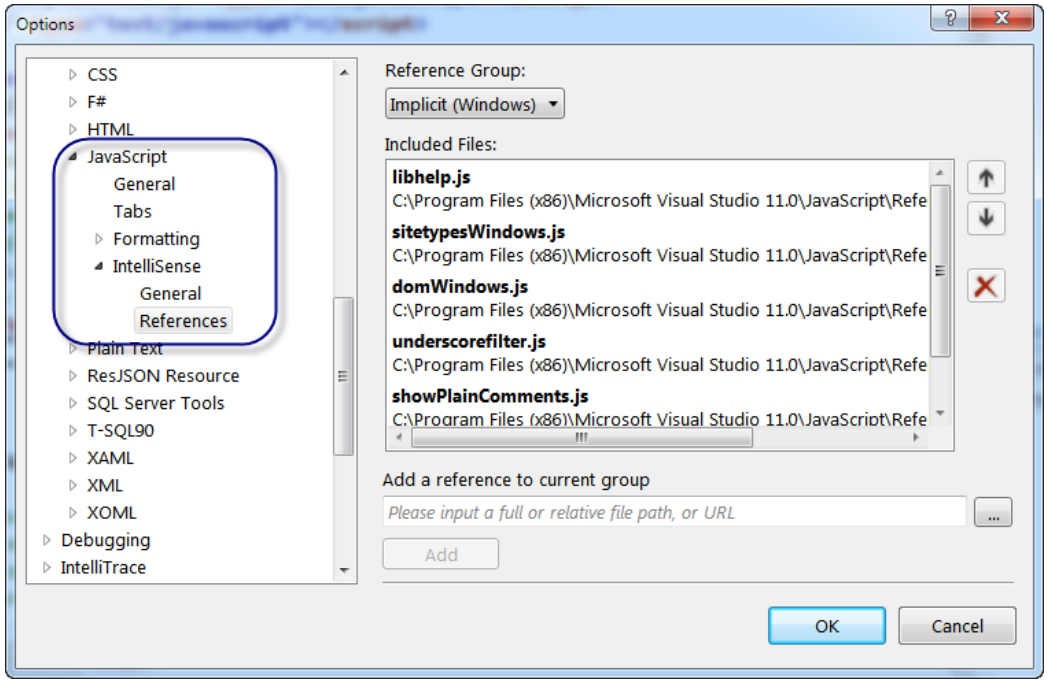


Fig. 17: Configuración de referencias externas para el editor de JavaScript

Otra opción interesante relacionada con la edición es el comportamiento que permite mostrar los identificadores definidos por el usuario, y detectar cuáles son los que tienen valores *undefined*. Este comportamiento se ve en la figura 18, donde se encuentra la lista de identificadores que sí están definidos en el fichero activo, junto al aviso de que el argumento que supuestamente recibe la función (*canvas*), no es reconocido por el editor.

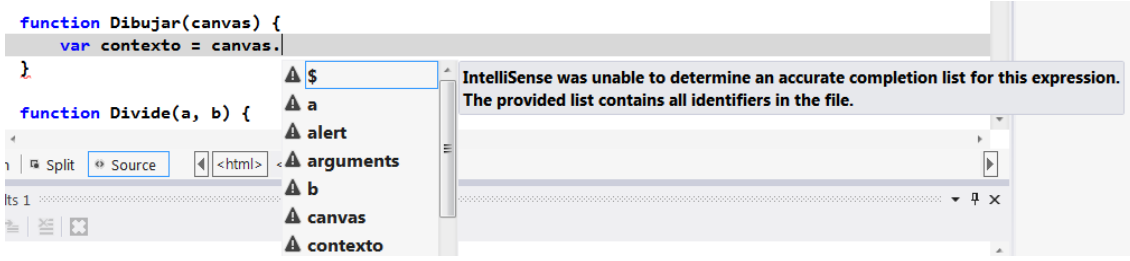


Fig. 18: Reconocimiento de identificadores definidos

En el caso de que necesitemos reconstruir los datos de *Intellisense*, disponemos igualmente del comando **CTL+SHIT+J** para hacerlo automáticamente. Hay muchas otras opciones que se han añadido para mejorar el soporte, como la precisión a la hora de mostrar información en las funciones con nombre, o el "*Script Loader*", que añade dinámicamente las referencias del contenido de otro archivo JavaScript al que se haga referencia en el código para su carga dinámica.

En la figura 19 podemos ver cómo el fichero "*Primero.js*" está cargado en la línea anterior, y la función definida en él -*A1(mensaje)*- aparece correctamente referenciada a partir de la siguiente línea de *script*:

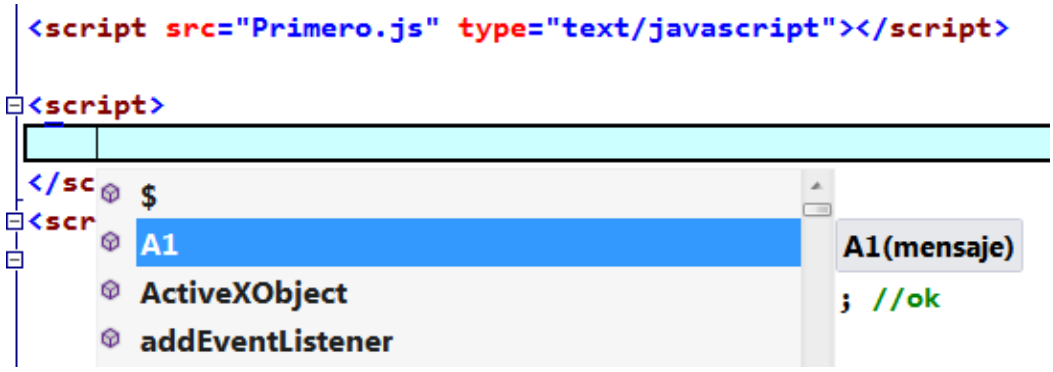


Fig. 19: "Script Loader" del editor de JavaScript en acción

Como una parte añadida de estas capacidades, también se puede depurar inmediatamente un archivo descargado de una fuente externa al programa (Recordemos que el Inspector de Páginas nos ofrecía la vista e identificación de elementos desde su panel "*Pila de Llamadas*").

Y a todo este paquete de mejoras hay que añadir las propias de los editores habituales de código ya presentes para otros lenguajes: resaltado de palabras, control y resaltado de las llaves de apertura/cierre, validación de expresiones regulares, con codificación de colores, soporte de la directiva "***use strict***" presente en ECMAScript 5, explorador del DOM, consola JavaScript y *snippets* de código configurados para este lenguaje.

Incluso podemos comentar otras mejoras vinculadas con este navegador,

como la consistencia con el chequeo sintáctico presente en IE10 (que será idéntica desde esta versión).

El editor gráfico de Visual Studio 2012/2013

Aprovechamos aquí para comentar una nueva funcionalidad perfeccionada en Visual Studio 2012/13: el editor de gráficos. Ahora es posible utilizar el editor junto con un paquete de herramientas de retoque gráfico que nos evitarán tener que utilizar programas externos para retoques sencillos, del tipo de los que muchas veces se necesitan en la edición Web.

Si seleccionamos un gráfico cualquiera, el nuevo editor nos ofrecerá ahora un conjunto de herramientas añadidas, como podemos ver en la figura 20.

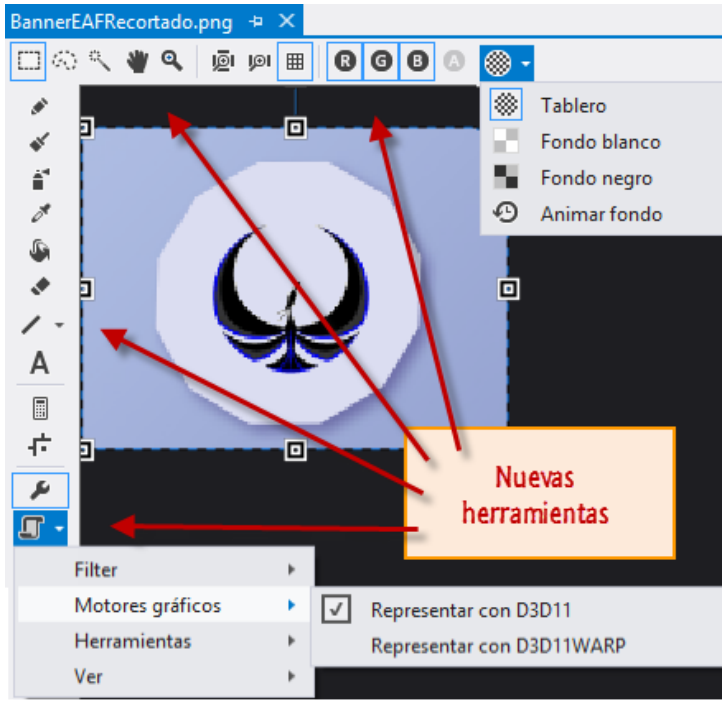


Fig. 20: Nuevas herramientas gráficas de edición en Visual Studio 2012/2013

Las herramientas de los navegadores

También aquí nos encontramos con muchas mejoras por parte de los agentes de usuario. A las consabidas ventanas adicionales de depuración (habitualmente con la tecla F12), se unen -en varios de ellos- otras opciones que ayudan a testar las aplicaciones y sitios Web desde distintos puntos de vista, y no solo el de la mera ejecución o interpretación correcta del código).

De hecho, cada nueva versión de un navegador incluye mejoras en su herramienta de depuración, y muchas de estas mejoras corren parejas con el soporte de los diversos sub-estándares de HTML5.

Algunos de estos avances los vemos en el apartado de las API de JavaScript y su soporte en depuración. Casi todos los navegadores actuales habilitan opciones para poder comprobar el funcionamiento en tiempo de ejecución de estas API (como los *Web Workers*, el almacenamiento local o de sesión, etc., facilitando no poco la implantación y la depuración finales).

A lo anterior hay que unir un conjunto cada vez más numeroso de simuladores (algunos asociados a los agentes de usuario y otros a diversos SDK de cada plataforma), que nos permiten evaluar cómo sería el comportamiento de nuestro sitio o aplicación en una plataforma distinta, en una tableta con orientaciones diversas, en un "Smartphone", etc.

El desarrollador web de hoy tiene que afrontar retos mucho mayores, por lo diverso del ecosistema de dispositivos y mecanismos que pueden acceder a internet y solicitar nuestra página, pero, paralelamente, también disponemos de más y mejores herramientas y de un nivel de

consenso mucho mayor a la hora de establecer cuál es la mejor solución a un problema de negocio concreto.

La herramienta de desarrolladores de IE11

La Herramienta de Desarrolladores de IE11 presenta muchísimas novedades, comenzando por la propia interfaz de usuario, que ahora obedece a otra disposición visual distinta, como vemos en la figura 21.

Está claro que el equipo de desarrollo se ha tomado muy en serio el soporte del estándar y la depuración en todas sus facetas, ya que no hay un aspecto que hayan pasado por alto. Desde el soporte mejorado de las características clásicas que ya estaban disponibles, hasta otras novedosas, relacionadas con el estándar.

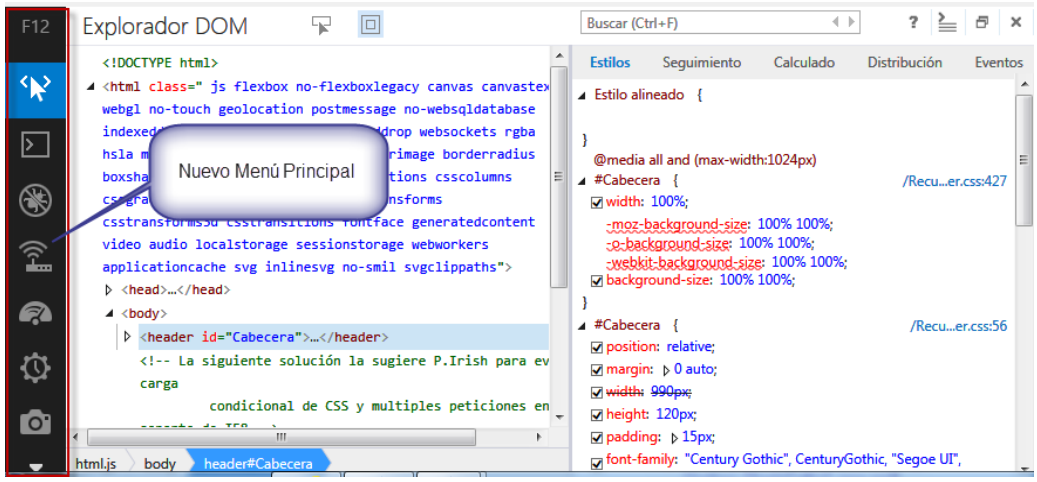


Fig. 21: Nueva Interfaz de la Herramienta de Desarrolladores de IE11

En el Explorador del DOM, podemos apreciar toda la jerarquía de elementos, identificar los estilos que están siendo aplicados, determinar el modelo de caja "calculado" para cada uno, y ver cuáles son los manejadores de evento de cualquier elemento, si es que los tiene.

En la opción de evaluar el modelo de caja "calculado", disponemos ahora de una nueva opción de filtro de estilos, para que resulte más fácil hacer el seguimiento de un estilo o familia de estilos e identificar problemas

potenciales. (Ver fig. 22)

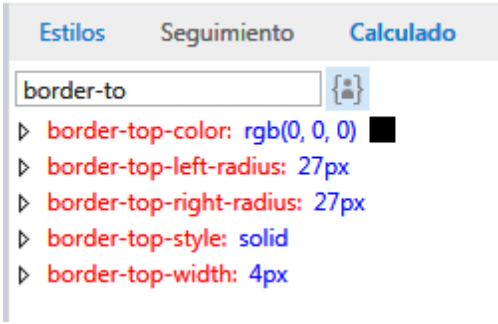


Fig. 22: la nueva opción de filtro de estilos calculados

También resulta de gran utilidad en depuración, como veremos más adelante, la opción de identificar los manejadores de evento aplicables a un estilo dado, como muestra la demo oficial de IE11 en el sitio http://ie.microsoft.com/testdrive y vemos en la Fig. 23:

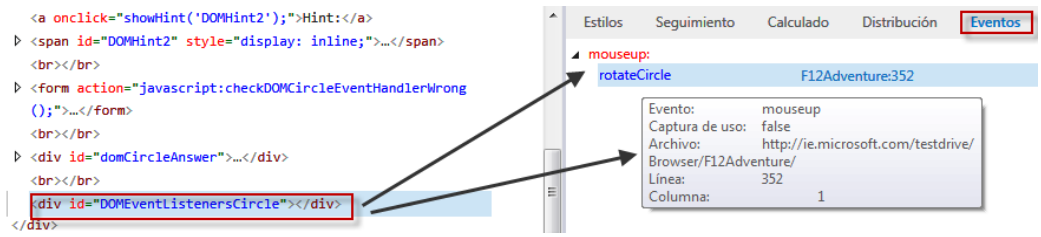


Fig.23. Explorador del DOM mostrando los eventos asociados a un elemento concreto.

Otra de las opciones novedosas y útiles que nos ofrece la herramienta consiste en la posibilidad de modificar la posición de un elemento… ¡en tiempo de ejecución!

De esta forma, si –por ejemplo- queremos modificar la posición de un gráfico dentro de un mosaico con más gráficos, podemos identificar el elemento (botón derecho sobre él, y seleccionar la opción "*inspeccionar elemento*"), y –una vez identificado en el DOM- podemos arrastrarlo a otra posición cualquiera. El visor de la página se actualizará inmediatamente para mostrar la nueva ubicación. La fig. 24 muestra el

proceso de una manipulación de este tipo.

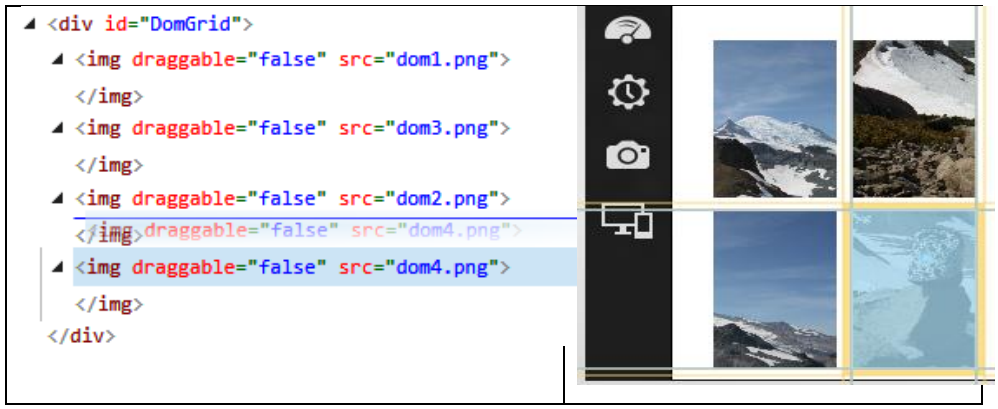


Fig. 24. Proceso de arrastrar y soltar un elemento en otra posición en IE11 (F12)

La Consola JavaScript, también se ha renovado considerablemente. Para empezar disponemos de *Intellisense*, dentro de la propia línea de comandos, como muestra la fig. 25:

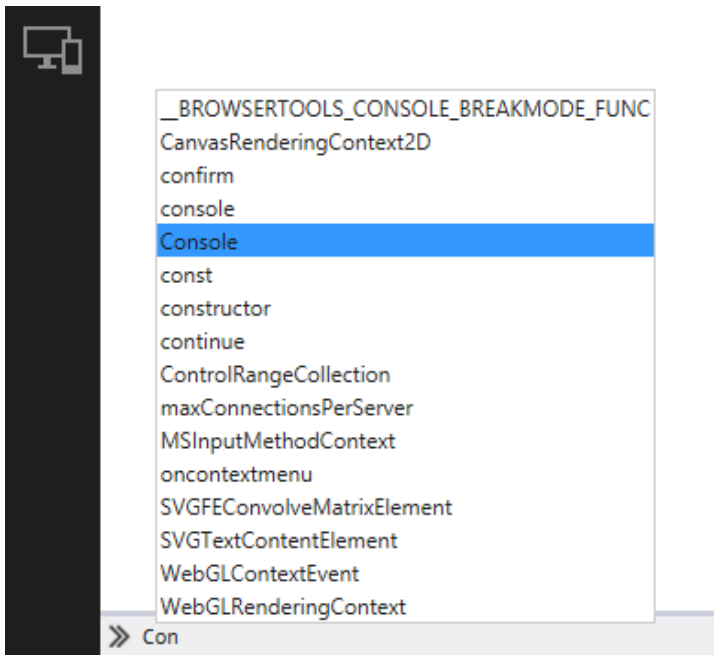


Fig. 25. Según vamos tecleando la consola va filtrando los posibles comandos a utilizar

También disponemos de entradas multi-línea, consolas disponibles para cada solapa de trabajo, así como un amplio conjunto de API para inspeccionar y probar el comportamiento de distintos elementos en tiempo de ejecución.

Por ejemplo, podemos inspeccionar en la consola tanto el estado de marcado de un elemento en un instante de la ejecución, como la representación de ese elemento como objeto, mediante los comandos dir y dirxml. Así, si tecleamos lo siguiente en la consola, veremos la salida que se corresponde con la figura 26.

```
var targetElement = document.querySelector(".console-
listing");
console.dir(targetElement);
console.dirxml(targetElement);
```

Fig. 26: Salida del código anterior por la consola de IE11

Y existen muchas otras API vinculadas con la consola que le aportan mucho valor desde el punto de vista de la depuración. Para una descripción detallada de estas opciones, ver el sitio web oficial http://ie.microsoft.com/testdrive/Browser/F12Adventure .

El Menú del Depurador es otro aspecto notablemente mejorado de esta herramienta. Ahora, podemos seleccionar cualquiera de los archivos JavaScript que utilice la solución, (tanto los propios de la aplicación o sitio, como los que pueda cargar dinámicamente) tal y como podemos ver en la fig. 27:

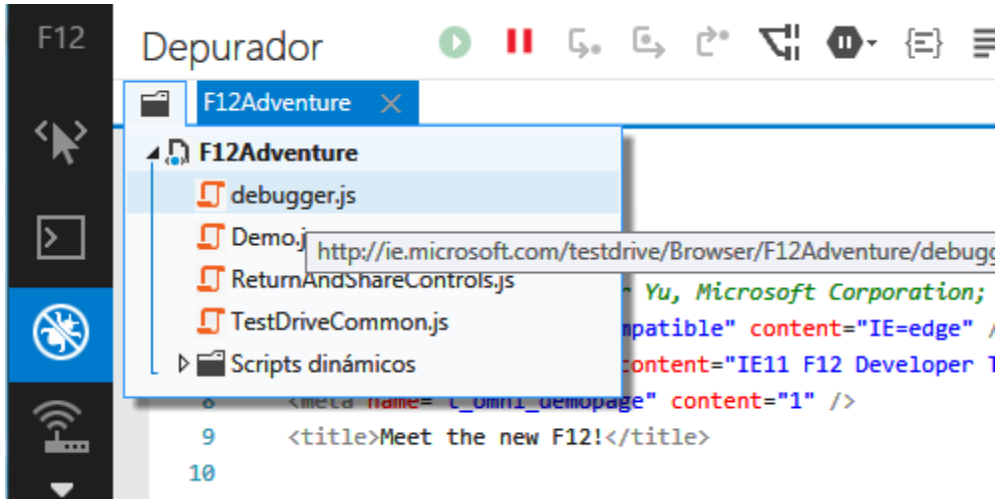


Fig. 27: El depurador de código JavaScript

Una vez realizada la carga del archivo, disponemos de la posibilidad de analizar el código fuente como lo haríamos con un código similar desde Visual Studio. Podemos establecer puntos de ruptura, disponemos de *Intellisense* y compleción de comandos, mecanismos de inspección y análisis predeterminado de todas las variables locales, Pila de Llamadas, y un largo etcétera. En la Figura 28 vemos estas opciones:

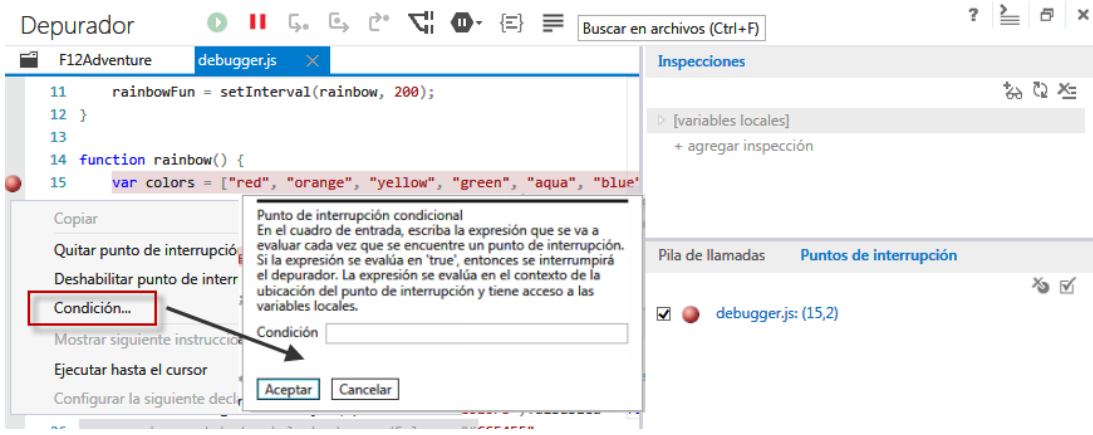


Fig. 28: El Depurador de IE11, mostrando un punto de interrupción y las sub-ventanas de Inspección y Pila de llamadas

De hecho, –de forma similar a Visual Studio- podemos ahora establecer puntos de ruptura condicionales, escribiendo expresiones que provocarán la interrupción del código si se evalúan a *true*, como aparece en la figura dentro del cuadro de diálogo complementario a la condición.

Además, una vez lanzado el proceso, podemos inspeccionar variables y valores de distintas ubicaciones, funciones, argumentos, etc. La Fig. 29 muestra esta situación en el contexto de ejecución real de la página indicada más arriba:

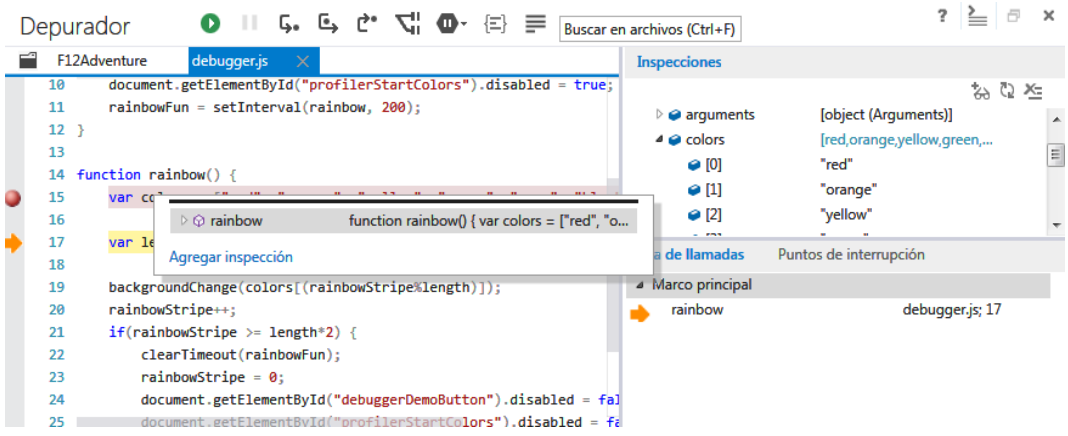


Fig. 29. El depurador mostrando valores de variables en tiempo de ejecución

El soporte de las nuevas API también ha sido cuidado y ahora en esta ventana observamos un botón que se corresponde con la opción de "*Interrumpir en nuevo Trabajador*", que saltará cuando se invoque mediante código la creación de un *Web Worker*.

Además, en la misma barra de tareas, disponemos de otras opciones útiles como las de cambiar el comportamiento de los puntos de ruptura, alternar entre vistas minimizadas y formateadas de código o alternar la inserción de puntos de rupturas en líneas muy largas.

La Herramienta Red

La siguiente opción, la herramienta "Red", también ha sido modificada

notablemente, para poder hacer un seguimiento del comportamiento de la aplicación en escenarios diversos.

Una vez que iniciamos la captura de tráfico de red mediante el botón podemos visualizar el proceso de una petición cualquiera, saber cómo ha concluido, cuál ha sido el tiempo empleado, el elemento iniciador del proceso, el método utilizado en la petición y el tipo de dato solicitado/transferido, junto a un pequeño gráfico explicativo que sirve de comparador de tiempos de procesamiento.

Como vemos en la figura 30, el menú "Red" nos muestra toda esta información en un recuadro y el menú de la parte superior, cuenta con las opciones de "Exportar el tráfico capturado" (que nos permitirá hacer comparativas del comportamiento de una en momentos distintos), "Borrar la Caché de Exploración", "Borrar Cookies", "Borrar entradas de navegación" (para repetir el proceso, por ejemplo) y "Eliminar todas las entradas". Esto podemos verlo en la primera de las vistas disponibles, la vista Resumen:

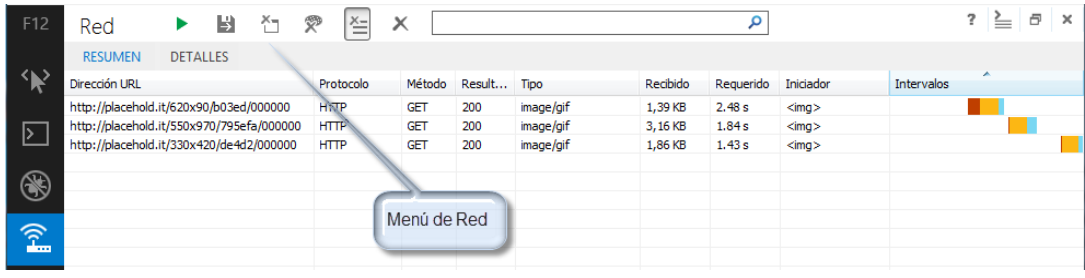


Fig. 30. El menú de Red, mostrando los resultados de una captura en modo global.

Como es lógico, también disponemos de una vista Detalles, que nos permite analizar una petición en concreto, visualizar su cabecera, el tipo de datos solicitado, el host, el tipo de codificación empleada, la conexión, etc. Y esto es válido para la petición y para la respuesta. Evidentemente, en una Web moderna conectada mediante servicios, esta información puede resultar crucial para ver el comportamiento de una petición cualquiera, especialmente en entornos AJAX.

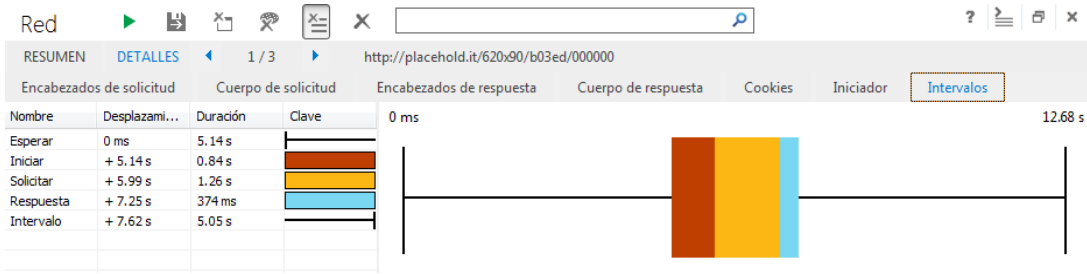


Fig. 31: La vista detalles del menu Red, mostrando la opción "Intervalos" de una de las descargas solicitadas.

Existe igualmente un menú llamado "*Capacidad de respuesta de la IU*", pero, en el momento de escribir estas líneas no se encuentra disponible más que para la versión IE11 de Windows 8, aunque lo estará para el resto de plataformas en la próxima actualización, según indican en su propio sitio Web.

La que sí se encuentra disponible es la opción del "*Generador de perfiles*", que permite recoger información de todo el proceso asociado a la carga de una página: cada función llamada, dónde lo ha sido, el tiempo transcurrido (con varios criterios), la URL (si corresponde), etc. La lista completa de valores es considerable, y además podemos configurar cuáles son los que queremos que aparezcan en el listado, como puede verse en la figura 32:

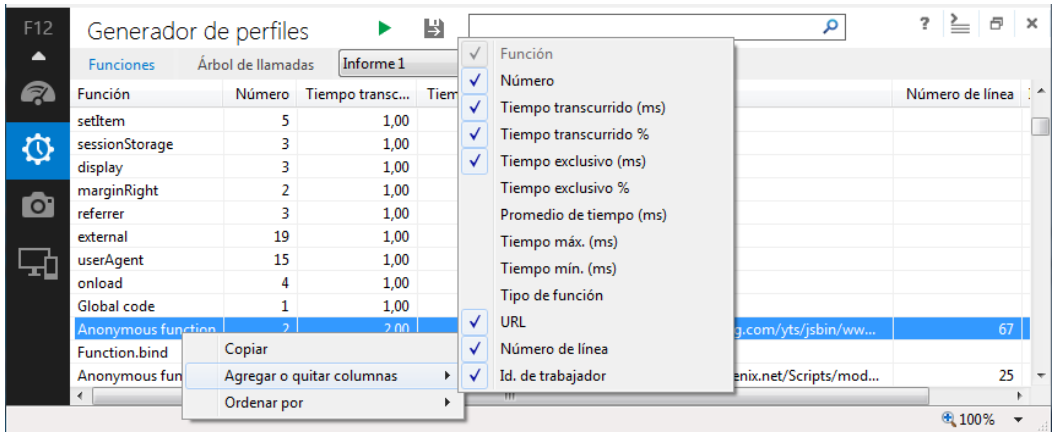


Fig. 32: Generador de Perfiles mostrando el resultado de una sesión de trabajo.

La opción de "*Memoria*", sirve para determinar el uso de la memoria que hace un elemento individual, o para buscar qué elemento está utilizando más memoria que el resto. Para ello, basta con iniciar una sesión de control de uso de la memoria y, en un momento dado, usar la opción "*Tomar instantánea de montón*". Una vez obtenida la instantánea (*snapshot*), podemos analizar los resultados, ordenarlos, buscar elementos por criterios de uso de memoria, etc.

El resultado que vemos en la figura 33 es la salida de una sesión de memoria sobre la demo predeterminada del sitio http://ie.testdrive indicado más arriba:

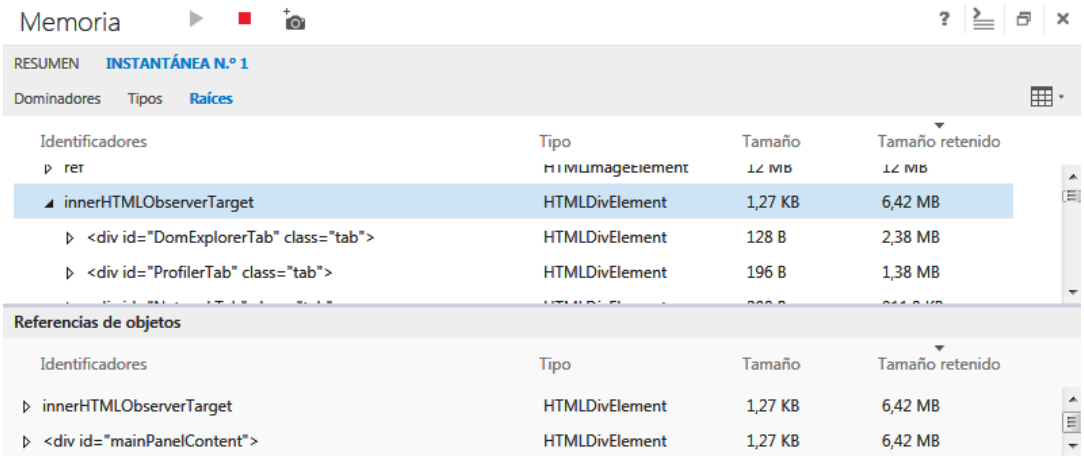


Fig. 33: Resultados de utilizar la herramienta de análisis de memoria en IE11

Finalmente, la opción de menú "Emulación", también ha dado un paso más a la hora de facilitar las pruebas de visualización en más de una plataforma y dispositivo.

Como se aprecia en la captura de pantalla correspondiente a la figura 34, disponemos de varias opciones de visualización, perfiles de explorador, cadenas de agentes de usuario, orientación y resolución de pantalla, e incluso, simulación de geo-localización:

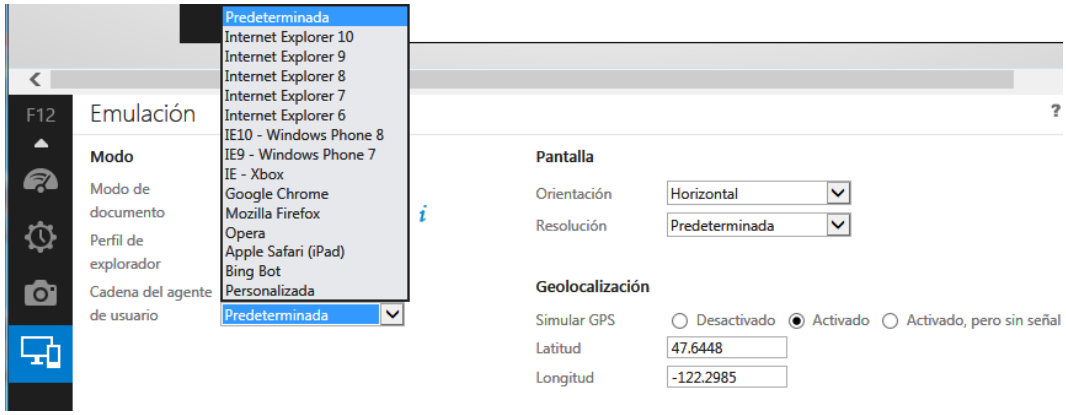


Fig. 34: La opción del Emulador, mostrando los distintos modos de emulación y el resto de opciones disponibles.

El sitio web oficial del producto indica[14], además, que "*IE11 Preview se ha optimizado para entrada táctil y es compatible con más estándares web que nunca, lo que facilita la interacción de los usuarios con los sitios en Windows, ofrece una seguridad de primer nivel e incluye herramientas de desarrollo que permiten la codificación entre exploradores (e incluso entre aplicaciones). Lo más destacado es lo siguiente:*

- *Los sitios funcionan mejor en IE11 Preview gracias a las nuevas mejoras de compatibilidad.*
- *Los sitios web y las aplicaciones de la Tienda Windows con JavaScript se han mejorado con experiencias 2D y 3D superrápidas y enriquecidas mediante **WebGL**, compatibilidad con valores altos de **ppp**[15] y nuevas características de Canvas.*
- *Las nuevas Herramientas de desarrollo F12 facilitan el desarrollo y la depuración de sitios entre exploradores y aplicaciones de la Tienda Windows con JavaScript.*

[14] http://msdn.microsoft.com/es-es/library/ie/bg182636(v=vs.85).aspx

[15] Puntos por pulgada (*Points per pixel*)

- *Los sitios reales se cargan más rápidamente con representación previa y captura previa, priorización de red y almacenamiento en caché de la navegación regresiva.*
- *Todo ello contribuye a crear una experiencia de exploración táctil dinámica y perfecta.*
- *La compatibilidad con iconos dinámicos, las actualizaciones de icono y las notificaciones de icono con RSS aumentan la personalización, la integración con Windows y la eficacia de los sitios anclados.*"

En resumen, una versión muy ambiciosa, con el objetivo de retomar una posición dominante en el contexto de los navegadores y un conjunto funcional y de desarrollo muy novedoso y actualizado, con excelente nivel de soporte de todas las características que forman parte de la "marca" HTML5.

Aunque es cierto que se trata de una versión *Preview*, resulta totalmente utilizable y es de esperar que la versión definitiva llegue junto a la versión final de Windows 8.1, o incluso antes. Ya solo falta que la política de actualizaciones, que, a diferencia de los navegadores de la competencia, ha estado un tanto anclada a las diversas plataformas (sin un desarrollo y actualización independientes), siga las últimas tendencias que la compañía promueve incluso en las herramientas de desarrollo y que se agrupan bajo el eslogan "*continuous deployment*", o actualización continua que están usando en este momento en Visual Studio 2012, por ejemplo.

FireBug

Muy popular y de excelentes prestaciones es -igualmente- *FireBug*, la herramienta del desarrollador de FireFox, a la que se accede por el menú o bien mediante la tecla F12. En principio, se trata de un complemento actualizable de muy fácil instalación, que puede configurarse para que aparezca siempre disponible en la parte inferior del navegador y también para que ofrezca un sistema de menús en la parte superior.

Firebug es un conjunto de herramientas con un entorno similar a la herramienta para desarrolladores de Internet Explorer (y otros navegadores), que –una vez habilitado-, nos ofrece la posibilidad de controlar los diversos aspectos de una página cargada según se muestra en la figura 35.

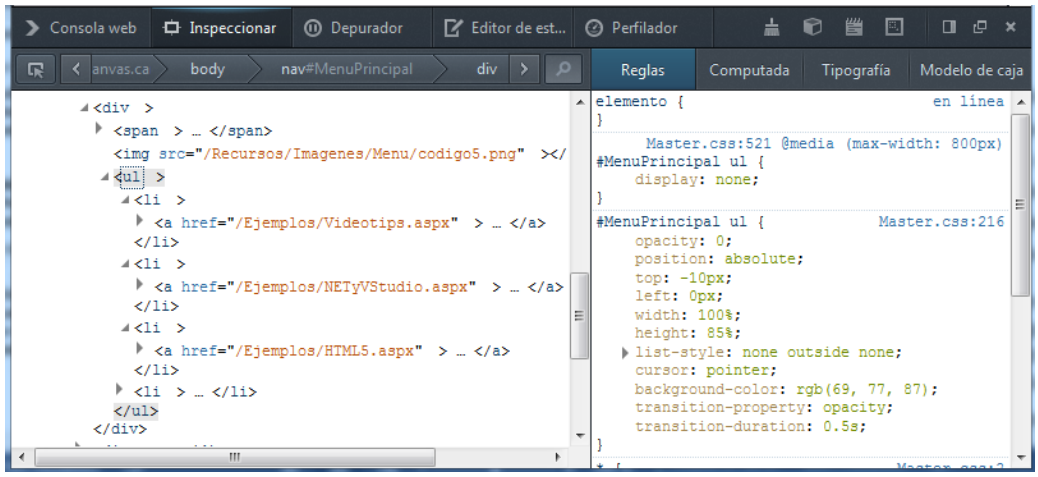


Fig. 35: Ventana inicial de la Herramienta para desarrolladores de FireFox

Como vemos, es similar al de IE (aunque se echa de menos alguna opción muy útil de aquel) y divide su funcionalidad en dos bloques de menús dependientes, con divisiones principales similares y con algún que otro añadido propio bastante cómodo, como es la opción  ("Seleccionar un elemento con el ratón"), para situarnos en cualquier elemento de la página, haciendo que –dinámicamente-*FireBug* localice el elemento en la ventana HTML y, una vez activado éste mediante otro clic, podamos usar la ventana derecha del complemento para analizar los diversos aspectos aplicables: estilos, JavaScript, ventana de DOM, tiempo de carga (si se trata de un recurso empotrado), etc. En el haber de la herramienta, cabe destacar algunas novedades que no encontramos en otras herramientas de navegadores, como el Visor de Tipografías utilizadas en la página activa, que nos muestra la Figura 36:

Fig. 36: Visor de tipografías

Por lo demás, también dispone de la mayoría de las opciones habituales, como el depurador de JavaScript, la Consola Web, el Generador de Perfiles de ejecución, etc.

Como novedad interesante, *Firebug* presenta también una alternativa al Emulador de Dispositivos que hemos visto para IE11, denominado "Vista de Diseño Adaptable", que nos permite definir cualquier modelo de visualización o utilizar el predeterminado, e irle modificando en anchura/altura según nos convenga para ver cuál es la respuesta dinámica de la página que analizamos. (Ver Fig. 37).

Fig. 37: La "Vista de diseño adaptable" de FireBug.

Otro aspecto –más curioso que práctico- que ya estaba presente en versiones anteriores de *Firebug,* es el "Visor 3D" de una página, que convierte la superficie de ésta en un elemento de 3 dimensiones (según los elementos en cascada que contengan) de forma que podemos manipular en los 3 ejes potenciales: rotarlo, desplazarlo, etc., obteniendo una vista similar a la que presenta la figura 38:

Fig. 38: Visor 3D mostrando la página principal de Danysoft

Por lo demás, nos encontramos con las opciones habituales de estas herramientas, si bien echamos de menos alguna de las opciones de gestión código JavaScript, especialmente, en lo relacionado con las nuevas API, como hemos visto en IE11, y veremos a continuación con la

herramienta del desarrollador de Google Chrome.

La herramienta de desarrollo de Google Chrome

Google Chrome lleva ya varias versiones soportando todos los recursos habituales de este tipo de herramientas y –desde hace más de un año- muchas de las opciones propias de manejo de API de JavaScript, como ***LocalStorage***, ***SessionStorage*** o ***IndexedDB***, accesibles dentro del menú *Recursos*, como podemos ver en la figura 39:

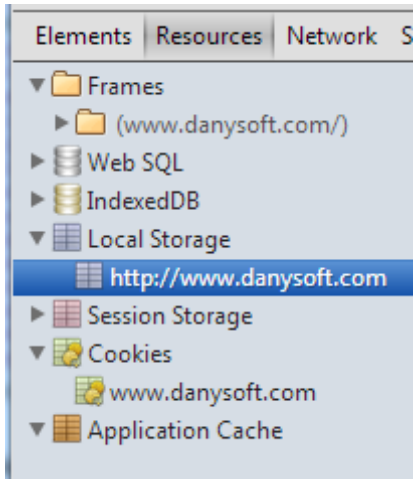


Fig. 39: El menú de Recursos de Crome, mostrando las opciones de soporte de las API Web SQL (obsoleta), IndexedDB, Local Storage y Session Storage.

Además, dispone de opciones muy detalladas para análisis de código, como por ejemplo, el visor de "*Excepciones del DOM*" (*DOM Exceptions*), como vemos en la figura 40, con el error resaltado:

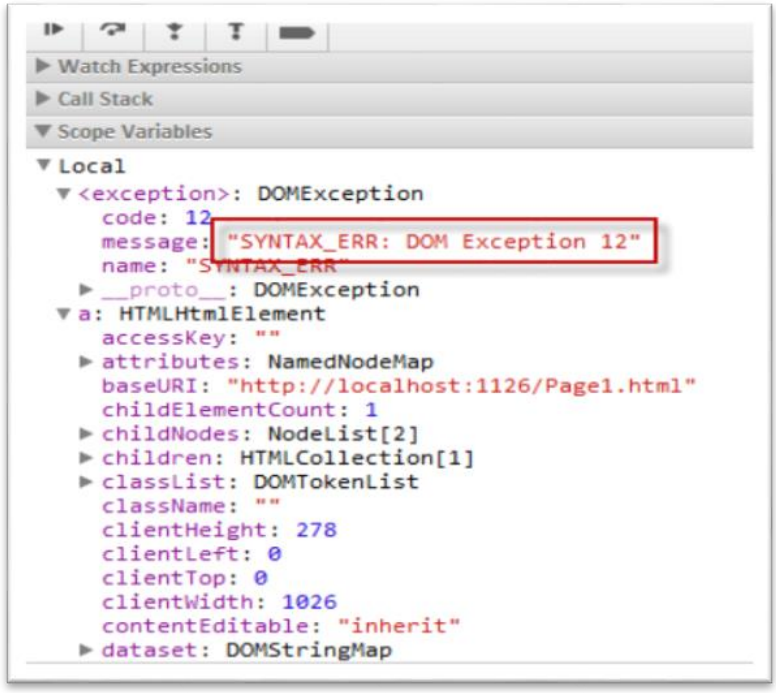


Fig. 40: Excepción del DOM en la ventana "Breakpoints" del depurador de Chrome.

De igual forma, cuenta con un mecanismo configurable de perfiles ("*Profiles*") para controlar el tiempo de CPU que consume cada recurso, pudiendo manejar independientemente 3 aspectos, de forma muy similar a como podemos hacer con IE11:

- Los que tienen que ver con *scripts*
- Los que están relacionados con hojas de estilo.
- La instantánea del Montón (*Heap*)

Se puede definir un perfil personalizado y la herramienta comienza un proceso de "grabación" en el que recolecta toda la información que se genere en tiempo de ejecución. Cuando la carga se completa, basta con volver a esta opción y comprobar el informe generado, como vemos en la figura siguiente con el informe producido mediante la tercera opción, que mostramos en la figura 41.

Observe el lector que podemos desglosar cada uno de los elementos analizados en sus componentes, que se mostrarán en la ventana inferior, consiguiendo un excelente grado de detalle.

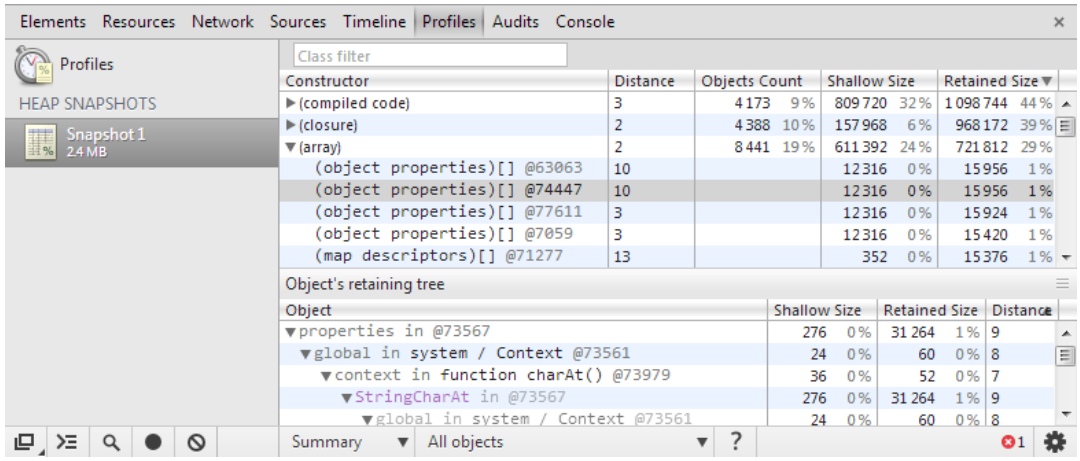


Fig. 41: la herramienta de análisis de Chrome, mostrando los resultados de una instantánea

Chrome también dispone de un mecanismo de auditorías para realizar el seguimiento de los recursos de red, así como el rendimiento de la página en el proceso de carga (Opción "*Audits*", del menú principal).

Los resultados pueden ser igualmente desglosados en componentes, y utilizados como una "guía de optimización del sitio", que nos permita mejorar los tiempos de respuesta o determinar posibles cuellos de botella de cualquier página, como vemos en este otro ejemplo que se muestra en la Fig. 42:

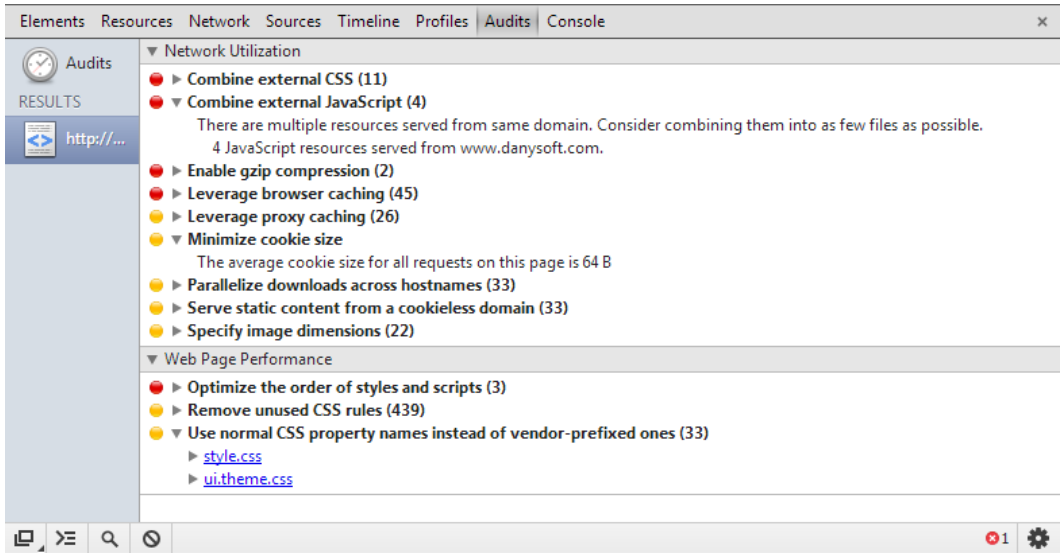


Fig. 42: Herramienta "Audits" de Chrome mostrando los resultados de una auditoría

Finalmente, cabe destacar igualmente, la opción "*Network*", que ofrece un análisis muy detallado de todo el proceso de tiempos de carga, con un estudio individual de cada uno de ellos, pudiendo visualizar los documentos, las hojas de estilo, las imágenes, los scripts, las llamadas AJAX (XHR), las fuentes externas y –como novedad interesante de las últimas versiones de Chrome, las llamadas generadas mediante la nueva API WebSockets que propone el estándar HTML5.

Podemos ver un resumen de estos aspectos en la Figura 43:

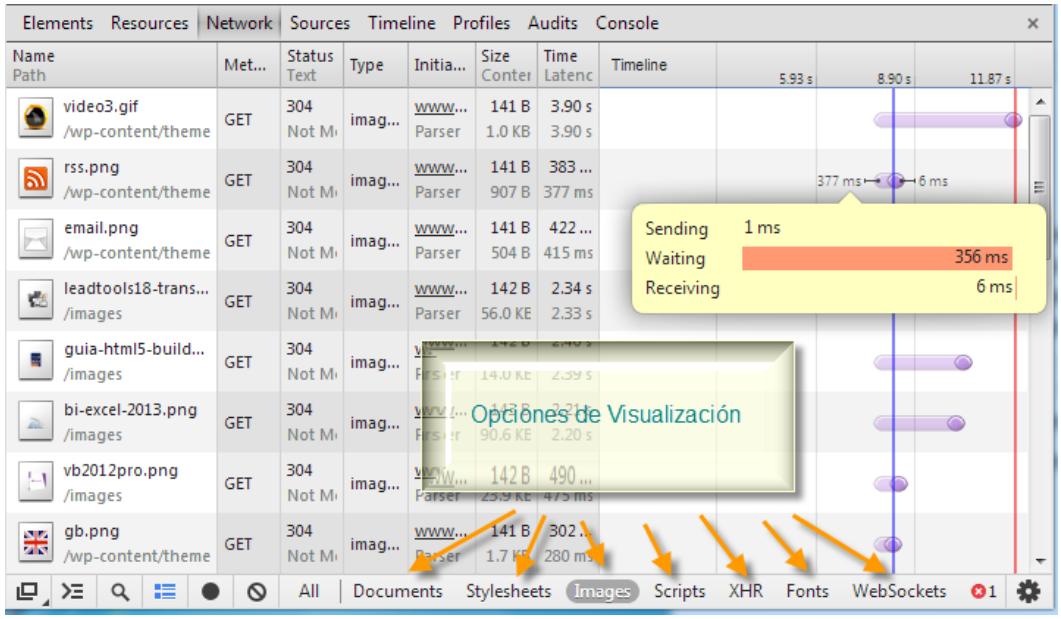


Fig. 43: Resultados del análisis de carga en la opción "Network"

Opera DraglonFly

DragonFly es el nombre que recibe la herramienta de desarrolladores del navegador Opera, que, en el momento de escribir estas líneas, se encuentra en su versión 12.16, y sigue la misma política de actualización permanente que están ahora siguiendo todos sus competidores.

En cuanto a su estructura, es muy similar a la de los otros navegadores comentados aquí, con las típicas opciones de navegación por el código fuente, depuración de scripts, Actividad de Red, Visor de Recursos, Ventana de Almacenaje, Gestor de Perfiles y Generador de Instantáneas. A su vez, dentro de estas opciones principales encontraremos las habituales de "*selección de elementos con un clic*", "*visor de estilos*", "*visor de la caja del elemento*", "*seguimiento de estilos*", etc. La figura 44 muestra el aspecto de *DragonFly* en funcionamiento, mostrando la ventana "Documento":

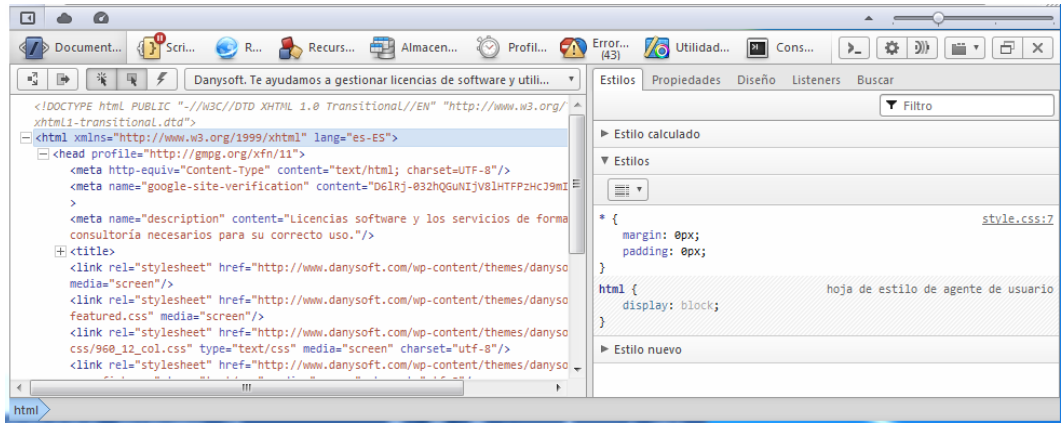


Fig. 44: Opera Dragonfly mostrando la ventana Documento con las opciones típicas.

En la Ventana Scripts podemos hacer un seguimiento de la ejecución del código JavaScript -igual que con los anteriores-, con un entorno muy completo que ofrece ventanas "*pop-up*" adicionales para controlar otros aspectos del código en ejecución, así como sub-ventanas (en la parte derecha) que muestran el estado, las interrupciones definidas en el código, las interrupciones de evento, y un opción de búsqueda para navegar rápidamente por el código fuente.

Podemos ver esta opción en funcionamiento en la figura 45:

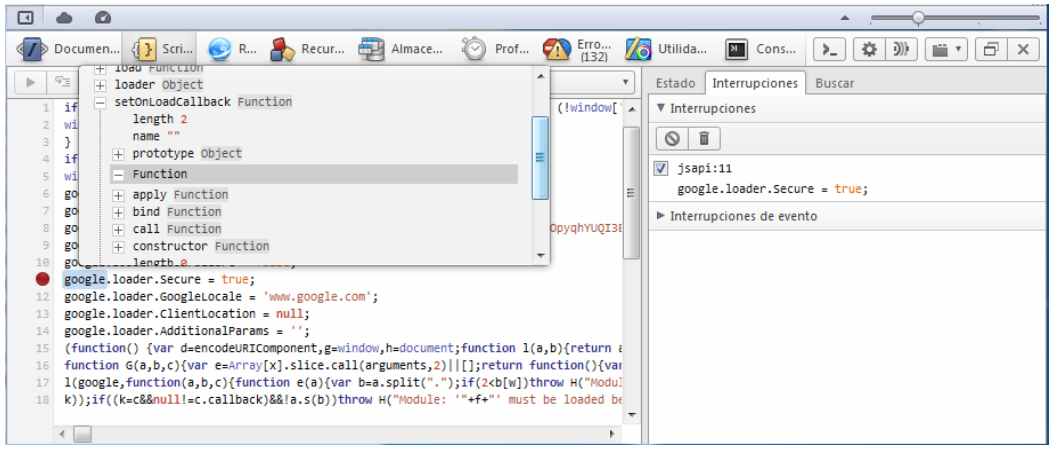


Fig. 45: Opción de depuración de scripts en Opera DragonFly

Por su parte, la ventana Red, ofrece igualmente un entorno muy completo, que no solo permite el análisis de los tiempos de carga, sino un desglose de cada uno de los recursos solicitados, pudiendo incluso realizar peticiones a un servidor directamente desde la herramienta, como vemos en la figura 46:

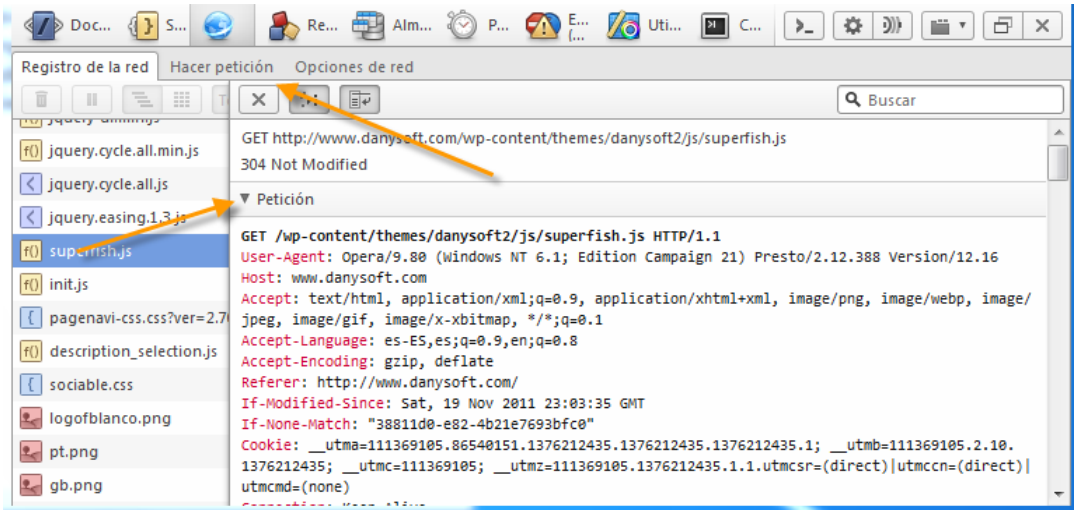


Fig. 46: La Ventana de Red, mostrando el resultado de una petición y otras solapas opcionales

Y así, podemos ir revisando el resto de opciones con la garantía de encontrar la mayor parte de las opciones disponibles en el resto de navegadores.

> **Nota**: el navegador Safari para Windows no ha sido actualizado desde la versión 5.1, dado que Apple se ha centrado en la versión para OS-X (versión 6), que dispone igualmente de un completo soporte de opciones de depuración, control de código fuente, y soporte de las nuevas opciones y API que presenta el estándar HTML5.

Otras herramientas: Fiddler

Fiddler es, sin duda, uno de los depuradores y analizadores de tráfico de internet (*sniffers*) más sofisticado, potente, especializado, extensible y configurable que podemos encontrar. Es una herramienta gratuita, que **Eric Lawrence**, ex *Program Manager* del equipo de desarrollo de Internet Explorer en Microsoft desarrolla y mantiene de forma independiente desde hace años.

En la actualidad, Eric trabaja exclusivamente en esta herramienta, desde que fue adquirida por la empresa **Telerik**, y el producto se encuentra actualmente en la versión 4.4.4.8 en su edición basada en .NET 4.0.

Vamos a comentar aquí lo más importante de su funcionamiento y características, aunque –por si misma- la herramienta podría contar con un capítulo completo.

Versiones de Fiddler y herramientas añadidas

Existen dos versiones de Fiddler, que se corresponden con la numeración Fiddler2 (siempre hay versiones beta disponibles), y otra realizada en .NET 4.0, de nombre Fiddler 4 (en beta muy avanzada y utilizable en el momento de escribir estas líneas). El sitio web16 de la herramienta dispone de información detallada de todo esto, así como de documentación descargable de forma separada.

Además del enorme número de opciones de depuración disponibles con esta utilidad, también cuenta con un Editor de Scripts (*Fiddler2 Script Editor*), que se instala en el mismo proceso que la herramienta principal y permite acceder a un Modelo de Objetos programable en JavaScript, para establecer condiciones personalizadas de depuración, en cualquier escenario u objeto imaginable. La figura siguiente muestra el editor de *Fiddler*.

[16] http://www.fiddler2.com/fiddler2/version.asp

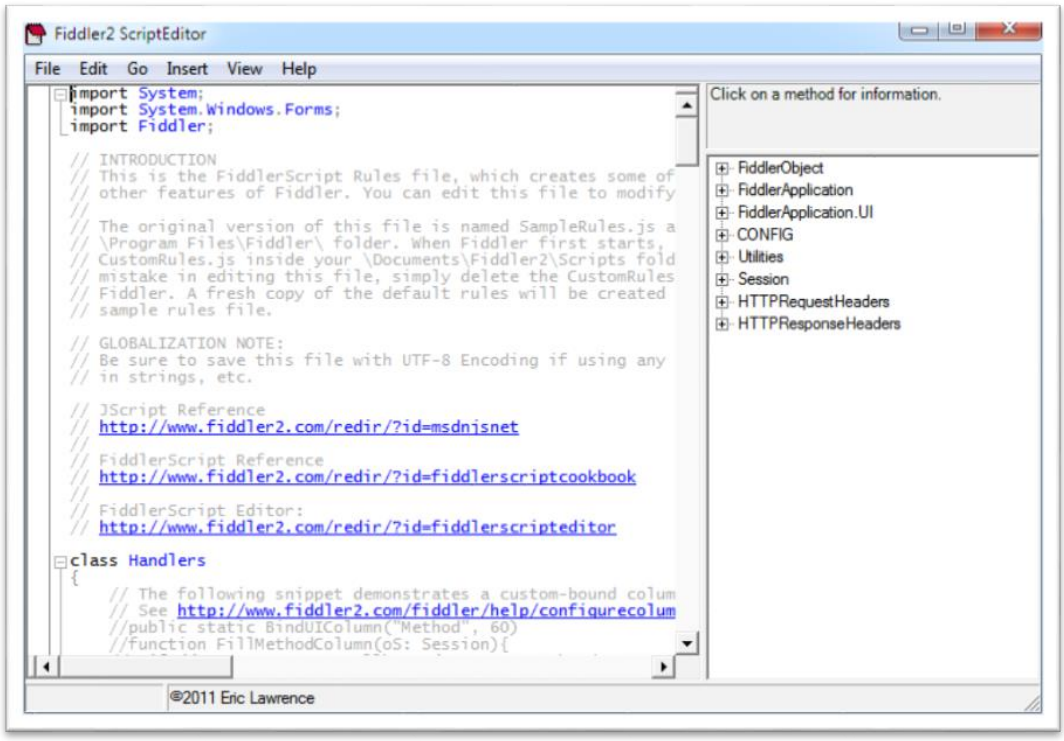


Fig. 47: El Editor de Scripts para el modelo de objetos de Fiddler.

Pocos podrán llegar tan lejos en el proceso de depuración, aparte de lo difícil que resulta encontrar un nivel personalización similar. Podemos obtener informes estadísticos de cualquier página y su tiempo de carga en cualquier navegador, e incluso configurar las opciones que preferimos que figuren en el informe y gráficos finales (Ver figura 48):

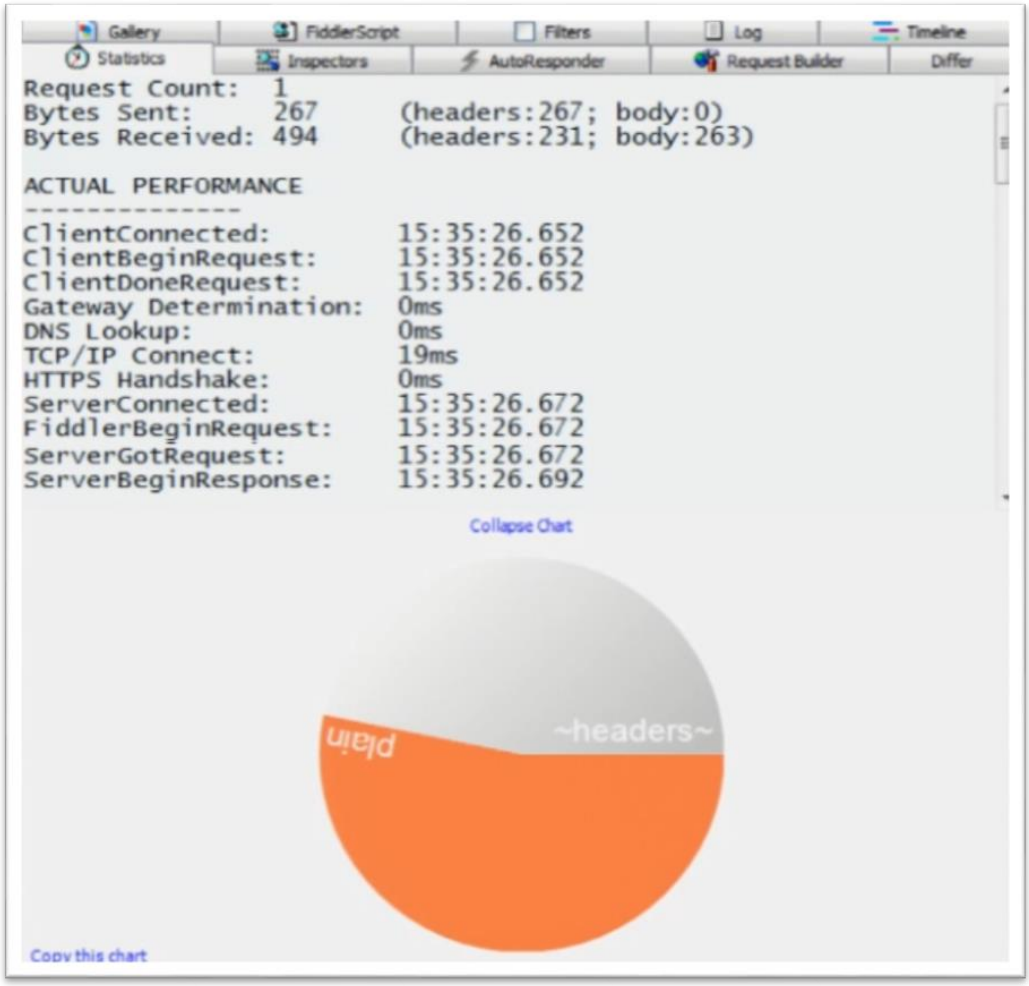


Fig. 48: Informe estadístico de Fiddler 2

Todo el tráfico puede ser almacenado y exportado/importado entre varios formatos compatibles para su análisis posterior en esta u otras herramientas, y todas las opciones valen, incluso para conexiones HTTPS de carácter seguro.

En este caso, Fiddler puede configurarse como fuente segura mediante un certificado especial que genera la herramienta al solicitar el análisis de este tipo de tráfico. La figura 49 muestra la ventana de creación de un certificado con este propósito.

Fig. 49: Creación de un certificado para habilitar la monitorización de tráfico seguro

Por si lo anterior fuera poco, la herramienta también dispone de varias extensiones que la hacen más potente aún. Entre ellas, hay que mencionar las siguientes:

- Un marcador sintáctico (*Syntax Highlighter*)
- Un formateador de código JavaScript, para mejorar la legibilidad (incluso de los archivos descargados dinámicamente)
- Una utilidad para configurar Fiddler para trabajar con aplicaciones "Metro Style" de Windows 8, la *AppContainer Loopback Utility*.
- Un generador de informes de rendimiento (*neXpert Performance Report Generator*), y…
- Una larga lista a la que hay que añadir extensiones de terceras partes, todas ellas también gratuitas e incluso programables mediante la interfaz ***IFiddlerExtension***.

En la figura adjunta vemos la herramienta de configuración de seguridad en Windows 8:

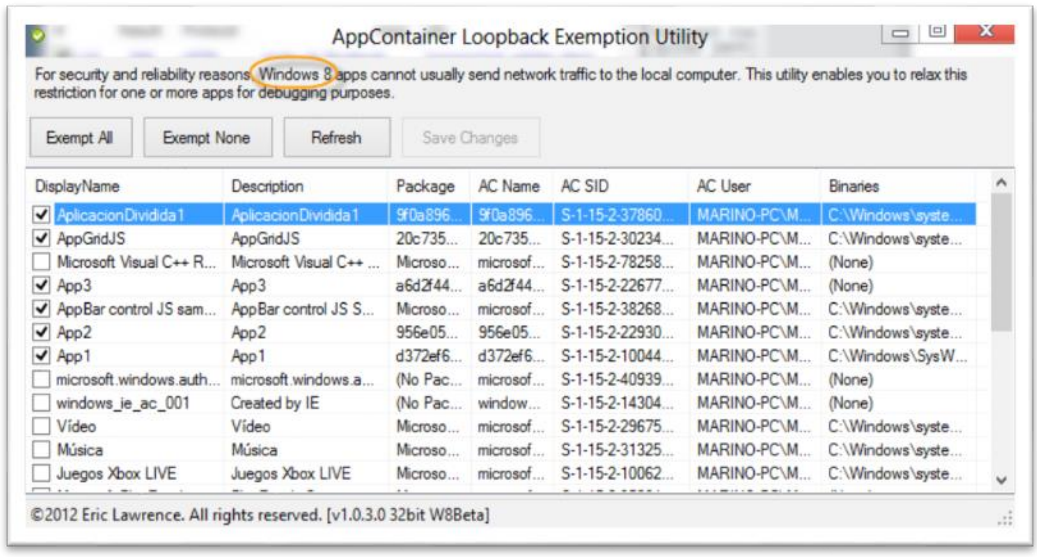


Fig. 50: Utilidad de configuración de control de tráfico en aplicaciones Windows 8

Arquitectura de Fiddler

Básicamente, Fiddler es un *proxy* que se ejecuta en la máquina local, y se sitúa de forma que puede capturar el tráfico activo. La figura siguiente muestra el esquema arquitectónico.

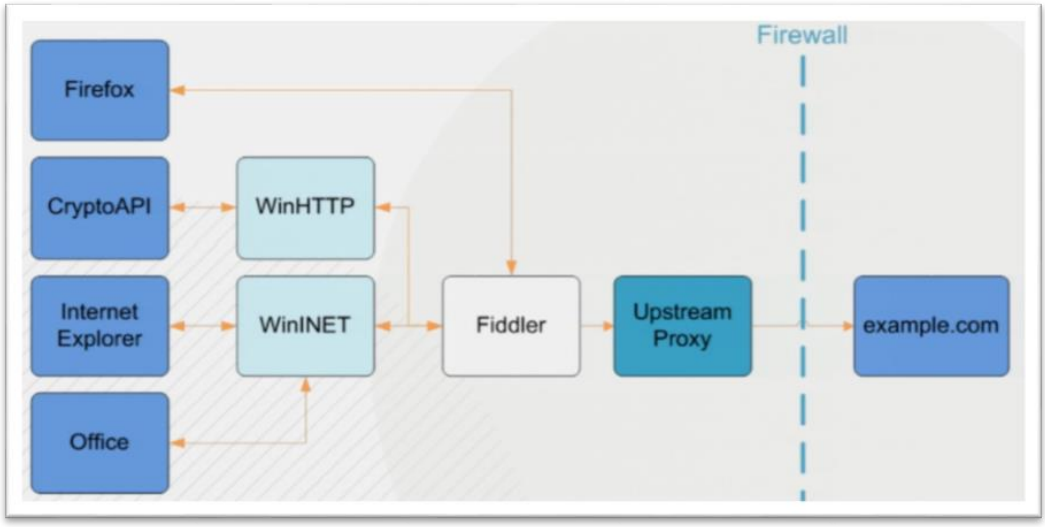


Fig. 51: Esquema del escenario de trabajo de Fiddler.

Una ventaja notable es que, como proxy, puede funcionar con cualquier cosa que lo soporte, como pueden ser aplicaciones, por lo que no se limita al tráfico de los navegadores.

La ventana principal, siempre presenta el tráfico generado por la actividad del usuario, permitiéndonos seleccionar sólo el proveniente de una fuente concreta, o incluso pudiendo elegir aquel que se esté generando exclusivamente como consecuencia de llamadas al servidor de Internet local (Http://localhost:), cosa de particular interés para los programadores, y una de las últimas opciones añadidas a Fiddler.

A todo lo anterior, hay que sumar que, a partir de las versiones de 2010, Fiddler puede trabajar de forma independiente del dispositivo, de manera que es posible capturar tráfico de otras plataformas (Unix, Mac), procesadores (ARM, etc.) y también dispositivos móviles (incluyendo, por supuesto, Windows Phone, como se indica en el gráfico siguiente:

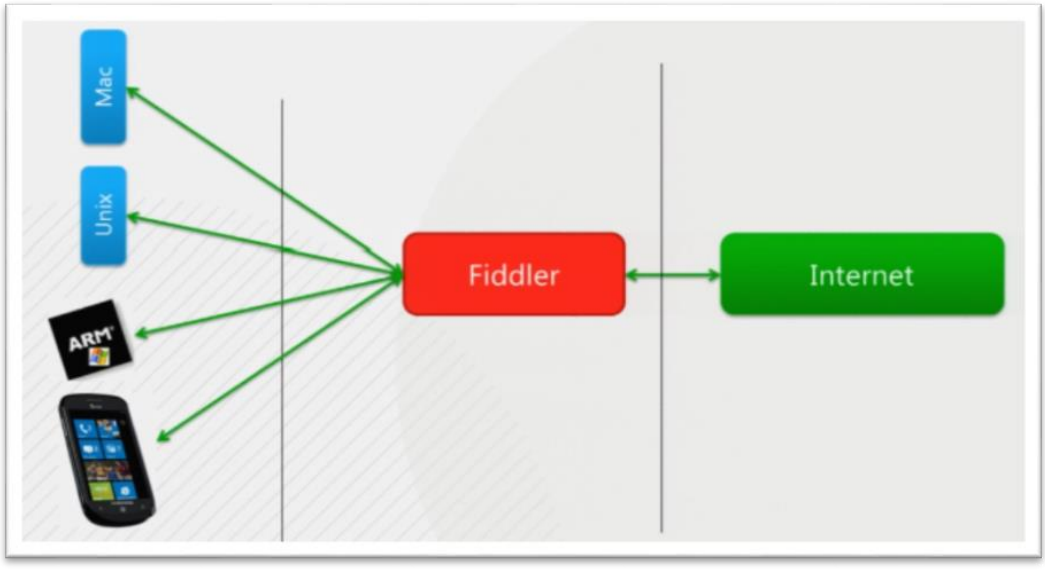


Fig. 52: Esquema del soporte de dispositivos y plataformas de Fiddler.

En el caso de los móviles, esto no significa que se pueda utilizar Fiddler desde el móvil, sino con el móvil, dado su carácter de proxy y la capacidad de trabajar en lo que se denomina "*Reverse Proxy Mode*" (Modo de *proxy*

inverso).

Generación de informes

Gracias a estas y otras opciones que descubrirá el lector al usarla, resulta sencillo instruir a Fiddler para conseguir realizar tareas únicas, como que nos guarde todas las imágenes de un sitio Web en una ubicación, o que exporte toda la información del tráfico a XML, filtrando solo las partes de nuestro interés, o que nos elabore un informe detallado de tiempos de carga de una página cualquiera.

En el apartado de nuevos "*Add-ons*", hay que citar el analizador "*HTML Analyzer Inspector*" (ver figura 53), que nos da un resumen (texto e imagen) de los porcentajes de utilización de cada recurso y el tiempo que han tardado en cargarse.

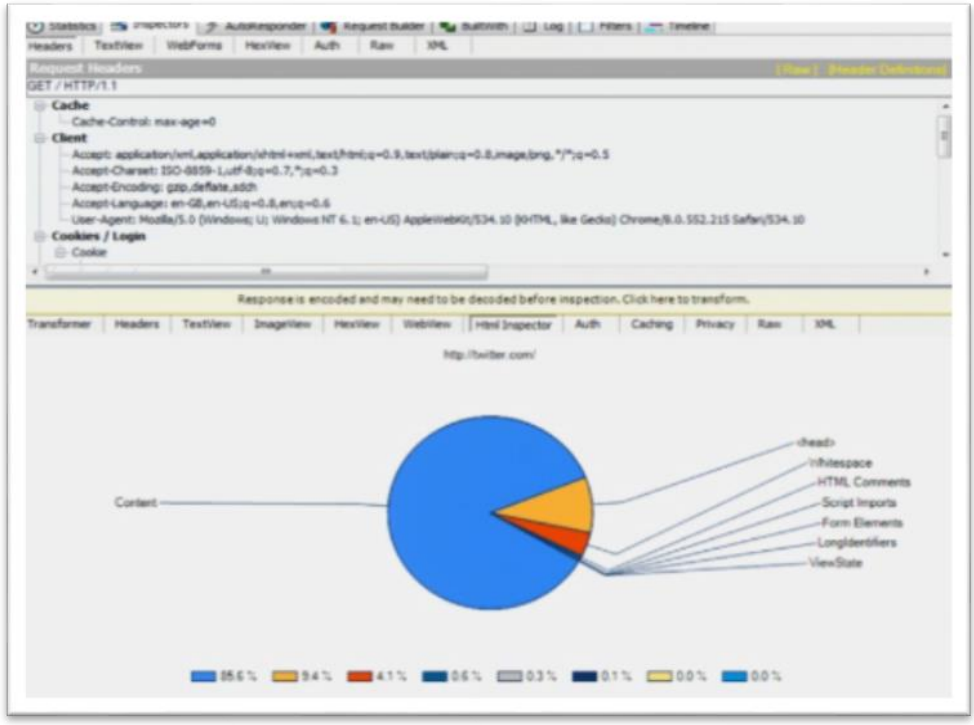


Fig. 53: HTML Analyzer Inspector de Fiddler

Por otro lado, y como complemento a la depuración con Visual Studio,

podemos usar Fiddler para controlar el tráfico generado por las aplicaciones que utilizan WCF (*Windows Communication Foundation*), gracias al complemento "*WCF Binary Inspector*", que vemos en funcionamiento en la figura 54.

Con él, se puede estudiar con detalle las peticiones realizadas a un servicio Web, así como las respuestas, pudiendo visualizar los contenidos en diversos formatos, e incluso utilizarla como herramienta para enviar datos personalizados al servicio, sin necesidad de programarlo de forma explícita.

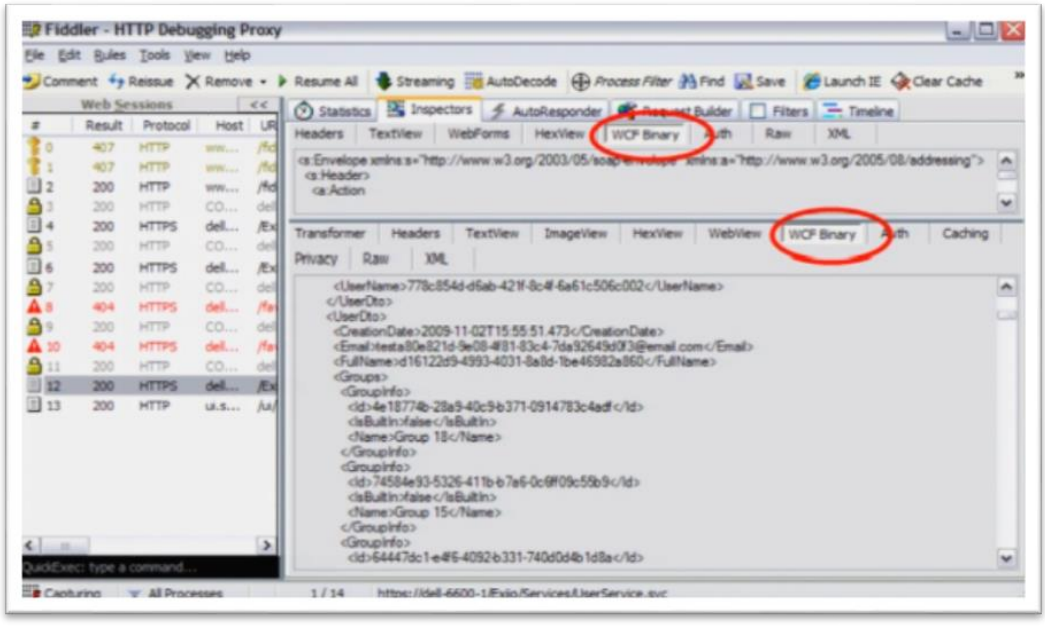


Fig. 54: WCF Binary Inspector en funcionamiento

El complemento descomprimirá el XML generado en el proceso y permitirá consultar el contenido resultante.

Otras extensiones de Fiddler

Además, como parte de las extensiones añadidas, disponemos de una herramienta de Auditoría de Seguridad, otra para pruebas de "*stress*"… y una larga lista que el lector encontrará en los enlaces indicados más

arriba, o en la documentación.

Finalmente, es posible empotrar el núcleo operativo de Fiddler, llamado "*FiddlerCore*", dentro de nuestras propias aplicaciones, sabiendo que el modelo de objetos de cara al programador es exactamente el mismo que el de la herramienta funcionando en la forma habitual. El esquema, podemos verlo en la figura 55:

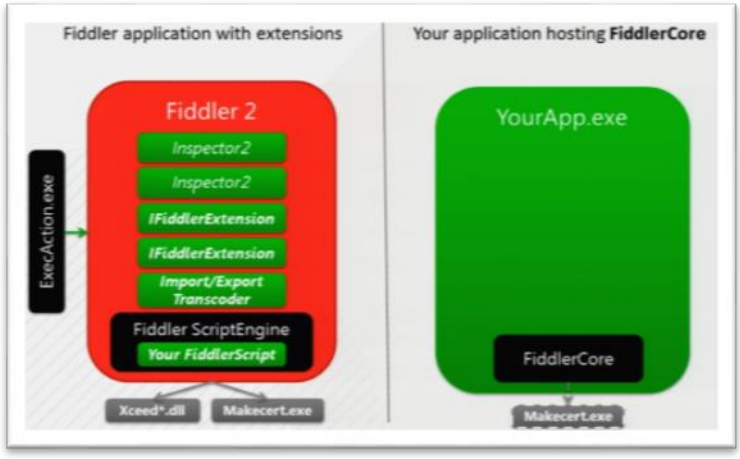


Fig. 55: EL modelo de objetos programable de Fiddler.

El modelo está disponible para .NET CLR 4.0, y soporta HTTPS y la captura de cualquier número de "*endpoints*" que se deseen monitorizar.

Fiddler como complemento de navegadores

Por último, Fiddler también puede instalarse como complemento de FireFox, para realizar sesiones de "*sniffing*" de tráfico desde este navegador. El complemento recibe el nombre de "FiddlerHook".

Para ello, basta con que instalemos la versión de Fiddler de escritorio y, al cargar una versión actual de otro navegador, nos aparecerá la opción de instalar el complemento de Fiddler, como podemos ver en la figura adjunta (también se puede buscar el complemento desde la página de complementos del sitio web de Fiddler):

Fig. 56: FiddlerHook: proceso de instalación en FireFox.

Una vez hecho esto, y habilitado el complemento, lo tendremos a nuestra disposición en el menú de "Herramientas" de FireFox, con lo que nos ahorramos el proceso de configuración manual o de reinicio del navegador.

En suma, una herramienta imprescindible para todo lo que signifique visualización del tráfico de la información, control de rendimiento, filtrado de canales de tráfico, etc. Y una opción muy a tener en cuenta para incluir capacidades de control de tráfico en nuestras propias aplicaciones.

Conclusión

Hemos visto el estado de finalización de los distintos estándares que forman parte de HTML 5, y también el soporte con que contamos en distintas herramientas, tanto de desarrollo, como las incluidas en los navegadores.

En el próximo capítulo, comenzaremos a estudiar las nuevas etiquetas

del lenguaje de marcas.

Referencias

- **IE Blog**. Sobre el desarrollo con Internet Explorer: http://blogs.msdn.com/b/ie/
- **MSDN Microsoft**, " Internet Explorer 11 Preview guide for developers", http://msdn.microsoft.com/library/ie/bg182636(v=vs.85)
- **MSDN Microsoft**, "Guía de Windows Internet Explorer 10 para desarrolladores", http://msdn.microsoft.com/library/hh673549.aspx
- **Sitio oficial de descarga de Visual Studio 2013:** http://www.microsoft.com/visualstudio/eng/2013-downloads
- **Known issues for Visual Studio 2013 Preview:** http://support.microsoft.com/kb/2848395/en-us
- **Descarga de Visual Studio 2012:** http://www.microsoft.com/visualstudio/11/en-us/downloads
- **Microsoft**, "What's new in Visual Studio 2012", http://www.microsoft.com/visualstudio/11/en-us/products
- **MSDN Microsoft**, "Novedades de RC de Visual Studio 2012", http://msdn.microsoft.com/library/bb386063(v=vs.110)
- **Fiddler**, sitio oficial: http://fiddler2.com
- **FireBug**, pagina oficial de descargas y documentación: http://getfirebug.com/
- **Chrome Developer Tools**: https://developers.google.com/chrome-developer-tools/?hl=es
- **Opera Dragonfly:** http://www.opera.com/dragonfly/
- **Apple**, "Safari Developer Tools", https://developer.apple.com/technologies/safari/developer-tools.html

Capítulo 02 | La sintaxis HTML 5

Como vimos en el primer capítulo, los borradores de documentación del estándar que produce la W3C se publican y revisan constantemente en su sitio Web, y ahí se define el propósito de estos trabajos, su alcance, su estado actual, etc.

De hecho, en lo referente a la sintaxis HTML, se publican realmente dos documentos diferentes: uno para las empresas que construyen agentes de usuario[17] y otro para desarrolladores[18]. A la fecha de escribir estas líneas, acaban de aparecer dos actualizaciones de ambos documentos, que puede el lector ver en las referencias a pie de página.

Pero hay algo más. Quien quiera bucear en los prolegómenos de estos documentos *on-line*, verá que existe una referencia a otro documento

[17] http://www.w3.org/TR/2013/NOTE-html-markup-20130528/

[18] http://www.w3.org/TR/2013/NOTE-html5-author-20130528/

más, que se publica con un botón que nos permite activar/desactivar estilos propios que muestran -o no- el contenido a los desarrolladores.

Pero lo más llamativo no es solo esto, sino el título del documento: ***HTML 5.1 Developer View***. (Ver figura adjunta). Vamos a ver si aclaramos el entuerto:

Fig. 1: El documento recomendado actualmente por la W3C en relación con HTML5

En realidad, estamos hablando ya de la siguiente versión. ¿Supone eso que HTML5 está terminado? Nada de eso. Solo que la especificación continúa con este nombre, y según palabras del propio documento, las referencias anteriores se mantienen "*por propósitos históricos*".

El documento oficial de HTML 5 se encuentra ya en su estado "*Candidate Recommendation*", y –consecuentemente- no son de esperar cambios en su contenido, por lo que cualquier creador de agentes de usuario puede utilizarlo como "Documento de Análisis", en palabras de algún analista Web.

No obstante hay que destacar el carácter dinámico que actualmente ha cobrado todo esto. Me refiero al hecho de que ya no parece existir una separación entre los creadores del estándar y los que lo implementan, sino que hay una colaboración cercana, y esto supone que el propio documento señale algunos de sus puntos como "*en peligro de desaparición, debido a su falta de implementación*".

Concretamente, el documento se refiere a:

- El API ***Application Cache***
- Etiqueta ***<dialog>***
- Etiquetas <***details***> *y* <***summary***>
- Atributo *color* en ***<input type=color>***
- Atributos de fecha/hora extendidos en <**input>**: ***<input type=datetime>, <input type=month>, <input type=week>, <input type=time>, <input type=datetime-local>***
- Etiqueta ***<output>***
- Atributo ***scoped*** en la etiqueta ***style***: *<style scoped>*
- Atributo ***seamless*** en la etiqueta ***<iframe>***: *<iframe seamless>*
- *Custom scheme y Content handlers (**registerProtocolHandler** y **registerContentHandler**)*
- Algoritmo *Outline*
- Mecanismo de navegación a elementos incluidos en una etiqueta ***<cite="">***

El primer caso, (la caché de aplicación) parece que no tendrá problemas para llegar al estado de recomendación final (ya está implementado en varios navegadores, como veremos en el último capítulo). Pero, del resto, sí que es posible que algunos no lleguen a figurar en el estándar (al menos en éste), ya que –efectivamente- no se aprecia una implementación significativa.

Los objetivos del documento actual del estándar establecen los siguientes puntos de trabajo:

1. Definir un único lenguaje llamado HTML 5, que puede ser expresado en sintaxis HTML y XML
2. Definir los modelos detallados de procesamiento para fomentar implementaciones interoperables
3. Mejorar el lenguaje de marcas de los documentos web

4. Introducir nuevas etiquetas y API para tecnologías emergentes, tales como las aplicaciones Web.

Disensiones en la WHATWG

En el verano de 2012 produjeron noticias algo desestabilizadoras para el estado del estándar, que se relacionan con la otra entidad implicada en la construcción: la **WHATWG** (que mencionamos en el primer capítulo), una entidad independiente, que había comenzado antes que la W3C algunos trabajos de estandarización, con una propuesta denominada "*Web Applications 1.0*", y la idea de promover las "aplicaciones para internet" en contraposición con los sitios Web.

Ante la coincidencia de objetivos, unió sus esfuerzos con los de la W3C, según los parámetros que hemos visto en el primer capítulo. La persona encargada de la coordinación de estas dos entidades era **Ian Hickson**, quien originalmente trabajaba para Netscape/Opera.

Pero, coincidiendo con el paso de Hickson a Google, éste declaró[19] que quieren adaptarse a los cambios e ir evolucionando la nueva propuesta, y no seguir con los que calificaba despectivamente como "venerables" métodos de la W3C. En su lugar, promueve un sistema propio, sin número de versión (solo HTML), "y añadir nuevas características según las necesidades e intereses lo aconsejen".

Ante esta postura, hay que preguntarse ¿necesidades e intereses de quién? Naturalmente, esto supone un golpe al proceso de unificación de la Web y sus aplicaciones, y vuelve a plantear la situación de un estándar "propietario", forzado para muchos por la mayor presencia del navegador Chrome en el mercado. Es pronto para saber las consecuencias y es de desear que los navegadores unifiquen sus criterios

[19] *Update on the relationship between the WHATWG HTML living standard and the W3C HTML5 specification*: http://lists.w3.org/Archives/Public/public-whatwg-archive/2012Jul/0119.html

de acuerdo con unos mismos principios básicos.

El marco de trabajo y los objetivos

El problema es que una modificación en algo tan fundamental como el lenguaje de la Web, tiene consecuencias a muy largo plazo y de una amplitud extraordinaria. **Douglas Crockford**, uno de los creadores de JavaScript y el ideólogo del formato JSON, afirmaba en el evento MIX de Microsoft que "*cualquier cambio en un estándar es un acto de violencia, y solo debe de admitirse como un mal necesario para conseguir un bien mayor*" (en alusión a ECMAScript 5).

Por otro lado, se imponían cambios en un modelo que tiene 12 años, y en un contexto (la Web) donde esos 12 años han originado muchas novedades que debían tener un reflejo en ese modelo. Y todo esto, teniendo en cuenta que no se podía romper la compatibilidad con los muchos millones de Webs existentes y aunque algunas etiquetas se declaran ahora como obsoletas, se asumen con el estado de "*conforming*" (o sea, todavía utilizables).

Compatibilidad hacia atrás

Y es que HTML 5 se define como "compatible hacia atrás". Persigue mantener el lenguaje en un formato relativamente simple, y prescinde de algunos elementos y atributos propios de HTML 4.01, especialmente de los relacionados con la presentación visual (etiquetas ***<acronym>, <big>, , <font>***, etc.), que se consideran separadas del borrador y pertenecen ahora al apartado de Hojas de Estilo (CSS3).

Los navegadores, sin embargo, deberán mantener la compatibilidad con los viejos elementos y atributos y por eso la especificación HTML 5 separa claramente los requisitos que deben cumplir los navegadores, de los indicados para los autores de páginas Web. Estos, deben utilizar las nuevas propuestas para fomentar el uso del estándar y la mejora de la

Web, pero los navegadores deberán de seguir soportando lo antiguo junto con lo nuevo.

La Tabla 1 recoge aquellos elementos declarados como no recomendables para los autores, pero que los navegadores deben seguir soportando, por razones de compatibilidad.

Etiqueta	Descripción
<acronym>	No soportado por HTML5
<applet>	No soportado por HTML5
<basefont>	No soportado por HTML5
<big>	No soportado por HTML5
** **	No soportado por HTML5
<dir>	No soportado por HTML5
<font>	No soportado por HTML5
<frame>	No soportado por HTML5
<frameset>	No soportado por HTML5
<noframes>	No soportado por HTML5
<s>	No soportado por HTML5
<strike>	No soportado por HTML5
<tt>	No soportado por HTML5
<u>	No soportado por HTML5
<xmp>	No soportado por HTML5

Tabla 1: Etiquetas obsoletas en HTML 5

Sintácticamente, es compatible con los estándares HTML 4 y XHTML1 y los documentos son siempre servidos con el tipo *mime* **text/html.** No olvidemos que uno de las principales causas del fracaso de XHTML 2.0, fue la ruptura completa que proponía respecto a los estándares anteriores.

La sintaxis general de HTML

El estándar indica, que puede utilizarse sintaxis HTML 4 o XML indistintamente para crear documentos HTML 5. A primera vista, la recomendación puede parecer confusa, pero no lo es tanto. HTML 4 y XML disponen de estructuras sintácticas bien definidas para indicar sus contenidos, pero el tipo *mime* de la primera es **text/html**, mientras que en el segundo caso es **application/xhtml+xml** o **application/xml**.

La primera instrucción de todo documento HTML 5, por lo tanto, debe indicar, en una u otra sintaxis, el tipo *mime*, y esto se realiza en HTML mediante la directiva **<!doctype>**. Para HTML 5, el código se reduce a lo siguiente:

```
<!doctype html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Documento simple</title>
</head>
<body>
    <p>Párrafo</p>
</body>
</html>
```

Llama la atención la sencillez de la declaración del tipo de documento (***doctype***), que en HTML 5 pasa a ser simplemente la primera línea del código anterior.

Otra novedad a este respecto es que HTML 5 define igualmente el tipo *mime* ***text/html-sandbox***, que se presenta como la opción en el caso de

que el contenido no se considere de confianza (esto es relativo al autor, y tiene connotaciones similares a la ejecución de un complemento Silverlight ejecutándose sin privilegios especiales).

En el caso de que se trate de un documento con sintaxis XML, la directiva inicial queda sustituida por la declaración "estándar" de XML, de forma que el mismo documento anterior, tendría el siguiente aspecto:

```
<?xml version="1.0" encoding="UTF-8">
<html xmlns="http://www.w3.org/1999/xhtml">
   <head>
      <meta charset="UTF-8">
      <title>Documento simple</title>
   </head>
<body>
    <p>Párrafo</p>
</body>
</html>
```

Vemos que la cabecera es la propia de un documento XML que sigue la sintaxis formal definida en el estándar, y la etiqueta HTML contiene el atributo **xmlns** (*namespace*), haciendo referencia al sitio oficial donde se encuentra la especificación *XHTML Namespace* en la W3C.

El tipo de documento: DOCTYPE

La definición DOCTYPE ha establecido habitualmente el tipo de documento con que concordaba el texto que seguía a continuación, y esto es algo que no estaba previsto en las especificaciones. Pero cuando Microsoft estaba trabajando en IE5, se encontró con algo curioso. El navegador mejoraba tanto el soporte de los estándares, que las páginas anteriores ya no se procesaban como era esperado. Lo hacían correctamente (de acuerdo con la especificación), pero la gente lo esperaba *incorrectamente*, por decirlo así.

Lo que existía hasta ese momento, se basaba en las peculiaridades de los exploradores dominantes entonces, Internet Explorer 4 y Netscape 4. Especialmente, IE5 (en su versión para MAC) era revolucionario, pero para

preservar la compatibilidad hacia atrás hacía falta una solución novedosa. El propósito de la directiva **<DOCTYPE>** era "activar" las nuevas funcionalidades ofrecidas. Mozilla tuvo que incorporarlo en su siguiente versión del navegador para evitar que muchas de las nuevas páginas creadas según este nuevo patrón dejaran de interpretarse correctamente.

De esa forma, la directiva pasó a ser la forma de reconocer el contenido de un documento, y se ha estado utilizando habitualmente hasta ahora con un formato similar al siguiente:

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

El nuevo formato simplificado, elimina las largas definiciones y permite de forma simple, establecer la conformidad de cualquier documento con el estándar. El cambio se basa en que, en versiones anteriores, DOCTYPE se basaba en el estándar SGML, y requería de la presencia de una DTD (Definición de Tipo de Documento). Los navegadores ya soportan directamente esta sintaxis.

Codificación de Caracteres (*Encoding*)

Finalmente, conviene recordar que las formas posibles de codificación de caracteres: el método que permite convertir un carácter de un lenguaje natural (alfabeto o silabario) en un símbolo de otro sistema de representación, típicamente un número, aplicando normas o reglas de codificación. Las normas ASCII y UTF-8, son ejemplos habituales de ello.

En HTML 5 dispone de 3 formas de indicar la codificación[20]:

- Desde el punto de vista del transporte, usando la cabecera **`HTTP Content-Type`**.

[20] Según se indica en el documento "*HTML 5 differences from HTML 4*", disponible en http://www.w3.org/TR/2011/WD-html5-diff-20110525/

- Utilizar un carácter BOM (***Byte Order Mark***) al comienzo del fichero. Este carácter suministra una firma que establece la codificación.
- Usar un elemento **<META>** con un atributo **charset** que indique la codificación en los primeros 1024 bytes del documento. Por ejemplo, para indicar que queremos usar codificación UTF-8 podemos usar:

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

En lugar del anterior:

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

Aunque esta sintaxis siga estando soportada por las razones indicadas más arriba.

Dicho esto, vamos a pasar a estudiar los cambios realizados en esta versión de HTML, dividiéndolos por categorías: los elementos nuevos, y los atributos modificados desde HTML 4.01, o que han aparecido en esta versión, y completaremos la descripción con ejemplos que ayuden a ubicar el alcance y el propósito de las nuevas etiquetas.

HTML5: elementos nuevos y modificados

Bajo este apartado se recogen muchas sugerencias que los usuarios de todo el mundo habían apuntado como necesarias, aportando novedades en la semántica, estructura, multimedia, formatos de entrada, etc. En otras ocasiones, se ha optado por modificar o añadir funcionalidad a elementos ya existentes pero que no tenían demasiada utilización en la práctica.

Nuevas Etiquetas

La lista completa de elementos nuevos es la siguiente (por orden alfabético), acompañada de una breve descripción inicial:

- **article**: Artículo
- **aside**: Contenido tangencial
- **audio**: Audio
- **bdi**: Aislamiento bidireccional
- **canvas**: Lienzo para gráficos dinámicos
- **command**: Comando
- **command *type*=*command***: Comando con acción asociada
- **command *type*=*radio***: Selección única de una lista
- **command *type*=*checkbox***: Selección de un elemento con estado
- **datalist**: Opciones predefinidas para otros controles
- **details**: Control para solicitar información adicional
- **embed**: Integración para complementos de terceros
- **figcaption**: Texto de un elemento **figure**
- **figure**: Contenedor especial con texto opcional
- **footer**: Pie
- **header**: Cabecera
- **hgroup**: Grupo de encabezado
- **input *type*=*datetime***: Control de entrada de Fecha/Hora
- **input *type*=*datetime-local***: Control de entrada Fecha/Hora local
- **input *type*=*date***: Control de entrada de Fecha
- **input *type*=*month***: Control de entrada de un mes
- **input *type*=*time***: Control de entrada de una hora
- **input *type*=*week***: Control de entrada de una semana
- **input *type*=*number***: Control de entrada de un número
- **input *type*=*range***: Control de entrada numérica imprecisa
- **input *type*=*email*** –Control de entrada de un *e-mail*
- **input *type*=*url***: Control de entrada de una URL
- **input *type*=*search***: Campo de búsquedas
- **input *type*=*tel***: Control de entrada de un teléfono
- **input *type*=*color***: Control de entrada de un valor de color
- **keygen**: Control generador de un par de claves
- **mark**: Texto resaltado
- **main**: Contenedor principal de una página (*propuesto*)

- **meta *charset***: Declaración de la codificación de caracteres
- **meter**: Medidor escalar
- **nav**: Grupo de enlaces para navegación
- **output**: Resultado de un cálculo en un formulario
- **progress**: Indicador de progreso
- **rp**: Paréntesis en *ruby*
- **rt**: Texto en *ruby*
- **ruby**: Anotación en *ruby*
- **section**: Sección de un document
- **source**: Origen multimedia
- **summary**: Resumen, Título o Leyenda para el control **details**
- **time**: Fecha y/o Hora
- **track**: Pista suplementario para multimedia
- **video**: Vídeo
- **wbr**: Oportunidad de ruptura de línea.

El usuario debe tener en cuenta respecto a la etiqueta ***input*** que, bajo el mismo nombre, se albergan controles de IU distintos en su presentación y funcionamiento.

Todas estas palabras reservadas definen etiquetas nuevas (***tags***). Por supuesto, todas siguen disponiendo de sus correspondientes atributos que personalizan su comportamiento.

Y, al igual que sucedía en HTML4, cabe distinguir entre los atributos específicos (los que configuran el propósito principal de la etiqueta) y los generales, o globales, que forman parte de todas (o casi) las etiquetas, como ***style***.

Algunos de estos atributos han cambiado para añadir funcionalidad, mientras que, en otros casos, lo que cambia es el significado de la etiqueta: su semántica. Así sucede con el siguiente conjunto de etiquetas que aparecen como "modificadas" por el estándar.

Etiquetas modificadas

- **a** (Hipervínculo):
    - El atributo ***target*** ahora puede contener un nombre de

contexto de un navegador
    - Nuevo atributo ***media*** donde especificar tipos de dispositivo o medios de salida
- **b**: Se elimina su aspecto visual y se recalca su lado semántico, como texto resaltado
- **cite**: Solo permitido para citar títulos de obras (no personas)
- **hr**: Ruptura temática de carácter semántico
- **i**: Similar a ***b***: se recalca su lado semántico indicando texto que las convenciones tipográficas marcan en cursiva
- **menu**: Ahora representa una lista de comandos
- **meta**: cambios en sus atributos *http-equiv* y nuevo atributo *charset*
- **s**: Se elimina su aspecto visual y se recalca su lado semántico, como texto anteriormente válido pero que ya no lo es.
- **small:** Se elimina su aspecto visual y se recalca su lado semántico, en el sentido de "la letra pequeña de un documento"
- **u:** Texto que de forma habitual se presenta en negrita, sin connotación visual especial.

Hay algo que comentar respecto al nuevo sentido de las etiquetas ***b***, ***cite***, ***hr***, ***s***, ***small*** y ***u***. En todos los casos, anteriormente, representaban formas visuales de diferenciar texto. En la actualidad, todo el mecanismo de presentación visual reside en CSS3, por lo que pierde ese sentido (aunque sigue representándose visualmente con una diferencia).

En su lugar, se hace hincapié en el aspecto semántico (en este caso, de la importancia, el cambio de contexto, la necesidad de resaltar

Etiquetas obsoletas

Los elementos de esta sección no son para ser utilizados por los desarrolladores Web. Los agentes de usuario aún tendrán que apoyarlos y varias secciones en HTML definen cómo. Por ejemplo el elemento **isindex** está obsoleto pero el *Parser* de un navegador tendrá que tenerlo en cuenta.

Los siguientes elementos no están en HTML, ya que su efecto es solamente a la presentación y su función se maneja mejor por CSS:

- **basefont**
- **big**
- **center**
- **font**
- **strike**
- **tt**

Los siguientes elementos no están en HTML, porque su empleo daña la usabilidad y la accesibilidad:

- **frame**
- **frameset**
- **noframes**

Los siguientes elementos no están incluidos porque no han sido utilizados con frecuencia, han creado confusión, o su función puede ser manejada por otros elementos:

- **acronym** no se incluye porque ha creado mucha confusión. Los desarrolladores web deben usar ***abbr*** para las abreviaturas.
- **applet** ha quedado obsoleto en favor de ***object***.
- **isindex** puede ser sustituido por el uso de controles de formulario.
- **dir** ha quedado obsoleto en favor de ***ul***.

Por último, el elemento ***noscript*** sólo es conforme con la sintaxis HTML. No está permitido en la sintaxis XML. Esto se debe a que, con el fin de ocultarlo (no sólo visualmente, sino también evitar que se ejecuten secuencias de comandos, se apliquen las hojas de estilo, u otras acciones), el analizador HTML comprende el contenido del elemento ***noscript*** como texto sin formato. Eso mismo no es posible con un analizador XML.

Cambios genéricos para todos los elementos: Atributos globales

Dentro de este último apartado de atributos globales, la tabla siguiente contiene el listado de los nuevos atributos globales definidos por el estándar.

Atributo	Descripción	Valores posibles
accesskey	Especifica un atajo que se puede usar para acceder al elemento	Cualquier cadena de caracteres, indicando las pulsaciones requeridas
class	Identificador utilizado para referirse a una clase predefinida en una Hoja de Estilo.	El nombre de la clase en la hoja CSS
contenteditable	Establece si el usuario puede o no editar el contenido.	• **true** • **false**
contextmenu	Establece un menú de contexto para el elemento.	El ID de un elemento menú en el DOM
dir	Especifica la dirección del texto. (Derecha a izquierda, o izquierda a derecha)	• **ltr** • **rtl**
draggable	Indica si el elemento se puede arrastrar.	• **true** • **false** • **auto**
hidden	Indica que un elemento ya no es relevante o que no lo es todavía. El navegador no muestra los elementos que incluyen este atributo.	Es un valor *boolean*. Si está presente, su valor debe ser, o una cadena vacía, o un valor con el nombre canónico sin espacios • *[Cadena vacía]* • **hidden**

Id	Identificador global de documento. Usado por CSS y JavaScript.	El nombre del ID que se desea usar.
lang	Establece el lenguaje a utilizar.	• Un código válido de lenguaje tipo RFC 3066 • Cadena vacía
spellcheck	Especifica si el elemento admite corrección ortográfica.	• Cadena vacía • **true** • **false**
style	Especifica estilos para un elemento	La definición de estilo que queremos utilizar
tabindex	Ayuda a determinar el orden de tabulación cuando el usuario lo utiliza para desplazarse por los elementos de un documento.	Cualquier valor entero: 0, 1, 2, 7…
title	Indica un título a asociar con el elemento. Muchos navegadores mostrarán esta información cuando el usuario pase el cursor por encima del elemento.	Cualquier texto que será mostrado como una etiqueta flotante (***tooltip***)

Tabla 2: Atributos globales para todos los elementos de HTML 5

Al igual que sucede con los anteriores, muchos de estos atributos ya existían en la especificación anterior, pero no tenían el carácter global que tienen ahora, o sea, no formaban parte de todas las etiquetas.

De cualquier forma, para analizar con detalle el funcionamiento de estas novedades, nos resulta más adecuada una división por categorías.

Categorías de etiquetas

La importancia del grupo de etiquetas de la tabla 2, no es equivalente. Y por otra parte, tiene sentido dividirlas por grupos según su propósito, lo que ayuda bastante a su utilización práctica. De esta forma, vamos a ver aquellas que tienen que ver con los aspectos semánticos, multimedia, de ayuda a la navegación, indicadores visuales, etc.

Etiquetas estructurales o semánticas

Son aquellas que sirven para crear el armazón de una página Web. La diferencia ahora es que ese armazón puede contemplarse desde dos puntos de vista: el arquitectónico (el que establece los "ladrillos" o arquitectura de la página), y el semántico -totalmente nuevo- que sirve para un propósito similar, pero desde el punto de vista de la organización conceptual del contenido: de cómo se relacionan unos contenidos con otros.

En HTML 4.01, la arquitectura de las páginas se basaba principalmente en elementos **`<table>`** y **`<div>`**, etiquetas que no ofrecen ninguna indicación acerca del contenido, ni mucho menos de la forma en que esos contenidos se relacionan. Tampoco aportan ninguna otra pista que pudiera ayudar a los buscadores en su labor de indexación "inteligente" de las páginas. Aparte de esto, las webs de hoy contienen mucha información específica, que tiene un significado común (artículos, fechas, comentarios, entradas, calendarios, etc.).

Este problema, puede ser paliado en parte gracias a las etiquetas **section**, **article**, **aside**, **header**, **footer**, **nav** y **figure**, que hemos visto en la lista inicial. Estas etiquetas ofrecen una cobertura al problema de la "semántica estructural", y también existen otras como **time** o **meter**, que aportan su grano de arena en el segundo aspecto.

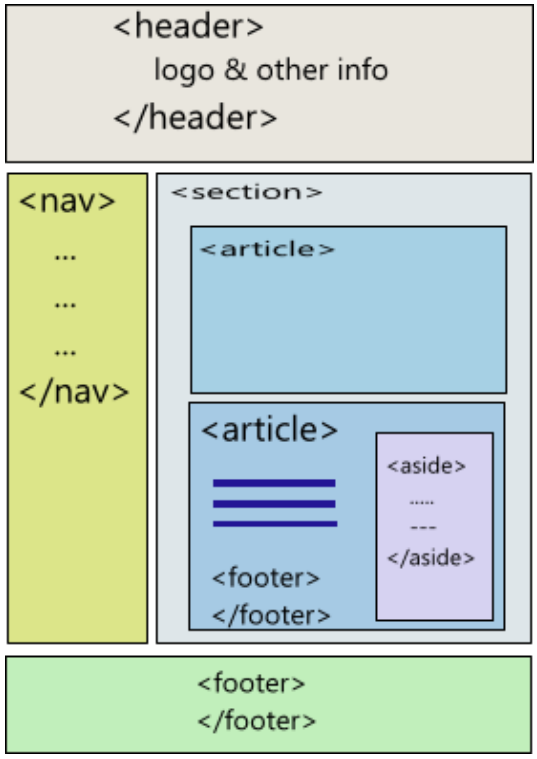


Fig.2: Esquema clásico de ubicación de elementos estructurales en un blog.

La fig. 2 representa un esquema típico de colocación de elementos en un blog o sitio web personal, pero es solo uno de los miles de esquemas válidos que son posibles. Vamos a revisar los fundamentos de cada nuevo elemento.

<section>

Es una de las etiquetas más difíciles de definir, pero, al día de hoy, existe consenso en comprenderlo como un "divisor semántico" exclusivamente. No debe entenderse como un contenedor (como **<article>**) sino como un marcador que señala divisiones que permiten después agrupar contenidos (y por tanto asociándoles un valor semántico). Puede decirse que su propósito es agrupar contenido que está relacionado por temática, fecha, autor u otro criterio adecuado.

La definición oficial indica que "**<section>** representa un documento genérico, una división lógica de contenidos o una sección de una aplicación". Y que "puede usarse junto a las etiquetas **h1**, **h2**, **h3**, **h4**, **h5**, y **h6** para indicar la estructura de un documento, pero, de la misma forma puede formar parte de otra etiqueta en la que tenga sentido dividir su estructura en secciones".

Volviendo a la documentación oficial, ejemplos de `<section>` podrían constituirlo capítulos, las diferentes solapas de un control ***Tab***, o el número de secciones numeradas de una tesis. Otro ejemplo perteneciente a un sitio Web común, podría dividir éste en secciones: *Introducción*, *Noticias* y *Contacto*. Y también nos da una pista nos da una pista sobre lo que debería de ser considerado como parte de una sección y lo que no: "El elemento `<section>` no es un elemento contenedor genérico. Cuando un elemento se necesita para fines de diseño o resulta conveniente para *scripting*, los autores pueden usar el elemento ** ** en su lugar. Una regla general es que el elemento `<section>` es apropiado sólo si el contenido del elemento figura explícitamente en el esquema del documento".

Por ejemplificar esta situación podemos pensar en un fragmento de código como el siguiente:

```
<section>
    <h1>Artículos</h1>
    <article>
        <header>
            <h2>Articulo1</h2>
            <p>Autor 1</p>
        </header>
        <section>
            <h3>Primera parte</h3>
        </section>
        <section>
            <h3>Segunda parte</h3>
        </section>
    </article>
</section>
```

```
<section>
    <h2>Comentarios</h2>
</section>
```

Donde hay dos secciones (Artículos y Comentarios), y –a su vez- podemos dividir el primer artículo en otras dos secciones. Visual Studio 2012/13, nos mostraría en la ventana de *Esquema de Documento* una estructura como la de la figura 3:

Fig. 3: Ventana de esquema de documento del código anterior en Visual Studio 2013

<*section*> y la noción de esquema de un documento

La estructura semántica de lo que se encuentra en el elemento **<body></body>**, cobra importancia en el estándar de cara a presentar la página al usuario. En HTML 4, las etiquetas **<h1>**…**<h6>** servían de base para esa estructura, que internamente es la que genera un esquema de documento (*Document Outline*).

Las relaciones entre los elementos de división (** **) y los de cabecera (la serie **h1**, **h2**…) es la que genera ese esquema de documento. No obstante, los elementos ** ** no son obligatorios para definir una

nueva sección, porque cualquier elemento de cabecera lo hace de forma implícita.

Expresando en términos de HTML, podríamos expresar esta idea de la siguiente forma: dado un documento HTML 4 con la estructura siguiente:

```
<body>
    <h1>Main Paragraph</h1>
    <p>This section starts here...this section goes on...</p>
    <h2>Second Level Paragraph</h2>
    <p>This is a subsection and... goes on...</p>
    <h2>Another Second Level Paragraph</h2>
    <h1> Information</h1>
</body>
```

El esquema de documento generado sería como sigue:

1. *Main Paragraph*
   a. *Second Level…*
   b. *Another Second Level*
2. *Information*

Esto presenta un buen número de problemas a la hora de evaluar la página para el algoritmo de esquematización: Si las etiquetas ** ** carecen del atributo **class** es complicado relacionarlas (su contenido solo de indexa, no se interpreta).

Además, la fusión de más de un documento, es propensa a errores. Cada sección es parte del esquema, y es complicado introducir excepciones que no sean consideradas como parte del esquema. También es fácil disponer de elementos que hagan referencia, no al documento, sino a todo el sitio.

En HTML 5, todo el contenido que esté incluido en la etiqueta **<body>** es parte de una sección. Por tanto, estas secciones pueden definirse de forma implícita o explícita. Los contenidos de las etiquetas **<body>**, **<section>**, **<article>**, **<aside>**, **<footer>**, **<header>**, y **<nav>**, definen secciones (divisiones) explícitas. Y además, cada sección puede disponer de su propia jerarquía de cabeceras, por lo que, incluso una subsección,

puede disponer de elementos **<h1>**, aunque la salida del intérprete visual reconozca correctamente esta circunstancia y presente la cabecera en una tipografía más pequeña.

También pueden existir secciones fuera del esquema del documento, y típicamente son aquellas que lo complementan, tales como **<aside>**, **<nav>**, **<header>** y **<footer>**.

<article>

Esta etiqueta presentaría unidades de contenido absolutamente independientes del resto. Pueden formar parte de una sección, o pueden contener varias secciones, sin que una cosa sea impedimento para la otra. Al tratarse de unidades independientes, se las podría trasladar en su integridad a otra página o a otra zona de la misma página sin pérdida de coherencia.

Por ejemplo, si publicamos un artículo, y admitimos una sección de comentarios de los lectores en su parte inferior, tendría sentido dividir el artículo en dos secciones, una para el artículo en sí, y otra para los comentarios.

El ejemplo anterior puede servir para ilustrar su uso habitual.

<aside>

Da soporte al contenido "paralelo" que tantas veces vemos en artículos de revistas, diarios y otras publicaciones. No está directamente relacionado con el texto, pero complementa al tema que se explica de forma que resulta una ayuda complementaria en la interpretación del texto.

Según la definición oficial, se trata de un "contenido tangencial" relacionado con el elemento principal al que complementa, pero que debe considerarse separado de él, en forma similar a como vemos habitualmente recuadros en una revista que explican conceptos

utilizados en el artículo principal, con el propósito de ayudar en su compresión. Utilizado en ese contexto, un elemento **<aside>** debería diferenciarse visualmente del contenido al que completa, mediante separaciones visuales y tipografía.

Por ejemplo, podríamos incluir una cita relacionada con un cierto texto, y crear una regla CSS para separarla del contexto de forma visual aunque permanezca dentro del bloque del artículo:

```
<article>
    <header>
        <h1>Titulo del Artículo</h1>
    </header>
    <p>Este texto pertenece al artículo cuyo título se
encuentra en el elemento superior a este. A continuación
aparece un elemento aside.
    </p>
    <aside>
        <q>Esto es una cita de otro autor</q>
    </aside>
    <p>Y aquí continúa el artículo. Se ha utilizado una
regla CSS sobre el elemento aside para separarlo
conceptual y visualmente del resto del artículo.
    </p>
</article>
```

El objetivo es conseguir una salida diferenciada como en la figura 4:

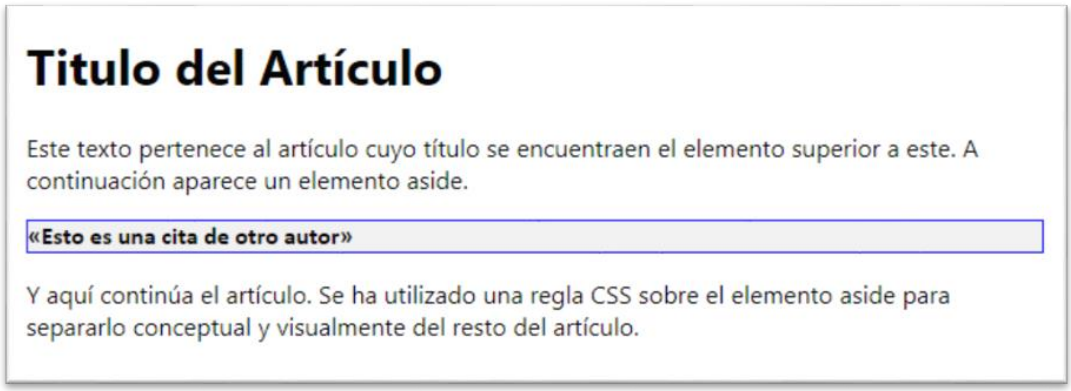


Fig. 4: Salida del código anterior en el que el elemento <aside> se diferencia visualmente de su contenedor, utilizando reglas CSS.

No debemos confundir esta utilización con las llamadas barras laterales (*sidebars*), que, la mayor parte del tiempo, son elementos fijos, sin relación con el contenido variable que se muestra a su lado.

<header>

De acuerdo a su significado, implica encabezamiento, presentación del contenido de un artículo o una sección, que puede incluir un mecanismo de navegación, el título y el autor o responsable de la sección/artículo, etc.

Su connotación semántica se encuentra en relación al elemento del que sirve de cabecera, ya que puede albergar contenido que está relacionado con la creación del contenido (autor, fecha, lugar, fuente, etc.) o la división en partes del mismo, permitiendo aportar enlaces adicionales o referencias.

Su uso puede ser el que hemos visto en ejemplos anteriores, si bien tiene sentido igualmente usarlo como cabecera de la propia página (Logo, Información corporativa, etc.).

<footer>

Es un pie de página, pero hay que entenderla en un sentido similar a

<header>, esto es, como una información complementaria al elemento al que sirve de pie. Puede contener enlaces complementarios, fecha de publicación, autor(es), y cualquier otra información relacionada que sea de relevancia. Incluso podría contener elementos **<section>, <nav>**, etc., si fuera apropiado. Lo que no debe de incluir es elementos **<header>** o más elementos **<footer>**. Se considera contenido fluido ("*flow content*").

Al igual que **<header>**, se entiende que complementan a *cualquier* elemento HTML y no solamente a los citados aquí. Por otra parte, no existe ningún inconveniente para que un elemento tenga varias etiquetas **<footer>**, si resulta adecuado.

El código siguiente es un ejemplo con un elemento **<section>** que contiene otro elemento **<article>**. Mientras éste contiene un solo elemento <footer>, la sección contiene dos:

```
<section>
    <h4>Contenido de la sección</h4>
    <article>
        Contenido del artículo
        <footer>
            Pie del artículo
        </footer>
    </article>
    <footer>
        Primer Pie de la sección (Enlaces...)
    </footer>
    <footer>
        Segundo Pie de la sección (Copyright...)
    </footer>
</section>
```

Por lo demás, tanto éste elemento como **<header>** insertan un retorno de carro a continuación de su elemento de cierre.

<nav>

Este elemento representa un fragmento de una página que enlaza con

otras páginas del mismo sitio o blog, o sea, una sección con enlaces de navegación internos. Esto no significa que otros enlaces propios del sitio deban de estar incluidos en un elemento **<nav>**, ya que los enlaces típicos de referencias en un artículo, o aquellos que indican páginas complementarias (*Condiciones legales*, *Copyright*, *Política de privacidad* etc.), no son candidatos a aparecer en esta etiqueta, pues un elemento **<footer>** sería suficiente y más apropiado. Debe aplicarse solo a los elementos que son parte del bloque de navegación principal.

El siguiente ejemplo, inspirado en código oficial publicado por la W3C en referencia con él, ilustra el uso de varias de estas etiquetas e indica cómo el uso de enlaces tiene un sentido diferente, según se trate de enlaces para desplazamiento por el sitio/aplicación o de tipo meramente descriptivo. Además de **<nav>**, el listado incluye el uso básico de las etiquetas **<header>, <footer>, <article>** y **<time>**. (Utilizamos un par de reglas de CSS para formatear una salida acorde con lo habitual en éste tipo de páginas).

```
<!doctype html>
<html>
<head>
    <title>Prototipo de Blog</title>
    <style>
        li { display: inline; margin: 0 3px; }
        #Contenido {
            font-size: x-large;
            background-color: #99CCFF;
        }
    </style>
</head>
<body>
    <header>
        <h1>Noticias de desarrollo</h1>
        <ul class="horizontal">
            <li><a href="Noticias.html">Noticias</a></li>
            <li><a href="Blog.html">Blog</a></li>
            <li><a href="Varios.html">Forums</a></li>
        </ul>
```

```
        <p>Última modificación:
            <time>11-06-2012</time>
        </p>
        <nav>
            <h1>Navegación</h1>
            <ul>
                <li><a href="Articulos.html">Índice de
artículos</a></li>
                <li><a href="Hoy.html">Noticias por
categorias</a></li>
                <li><a href="Tips&Tricks.html">Trucos y
sugerencias</a></li>
            </ul>
        </nav>
    </header>
    <article>
        <p id="Contenido">Reservado para el contenido de
la página
        </p>
    </article>
    <footer>
        <p style="font-size: x-small">Copyright ©2011 -
Danysoft</p>
        <p>
            <a href="acercade.html">Acerca de...</a>
            - <a href="politica.html">Política de
privacidad</a>
            - <a href="contacto.html">Contactar</a>
        </p>
    </footer>
</body>
</html>
```

Como puede verse, la salida correspondiente refleja una estructura predecible de los elementos que, aunque podría haberse estructurado de otra forma, resulta mucho más esclarecedora de cara a los robots de búsqueda. Hay que notar la presencia de varias etiquetas **<header>** y una etiqueta **<footer>** al final de la página.

Otro punto importante a observar es el distinto tipo de tamaño en la salida de dos elementos idénticos **<h1>**: el primero, a continuación de **<body>** tiene un tamaño mayor que el segundo, inmediatamente después de la etiqueta **<nav>**. El navegador ha interpretado correctamente que la importancia de la primera es superior a la de la segunda (respecto a la página en su totalidad).

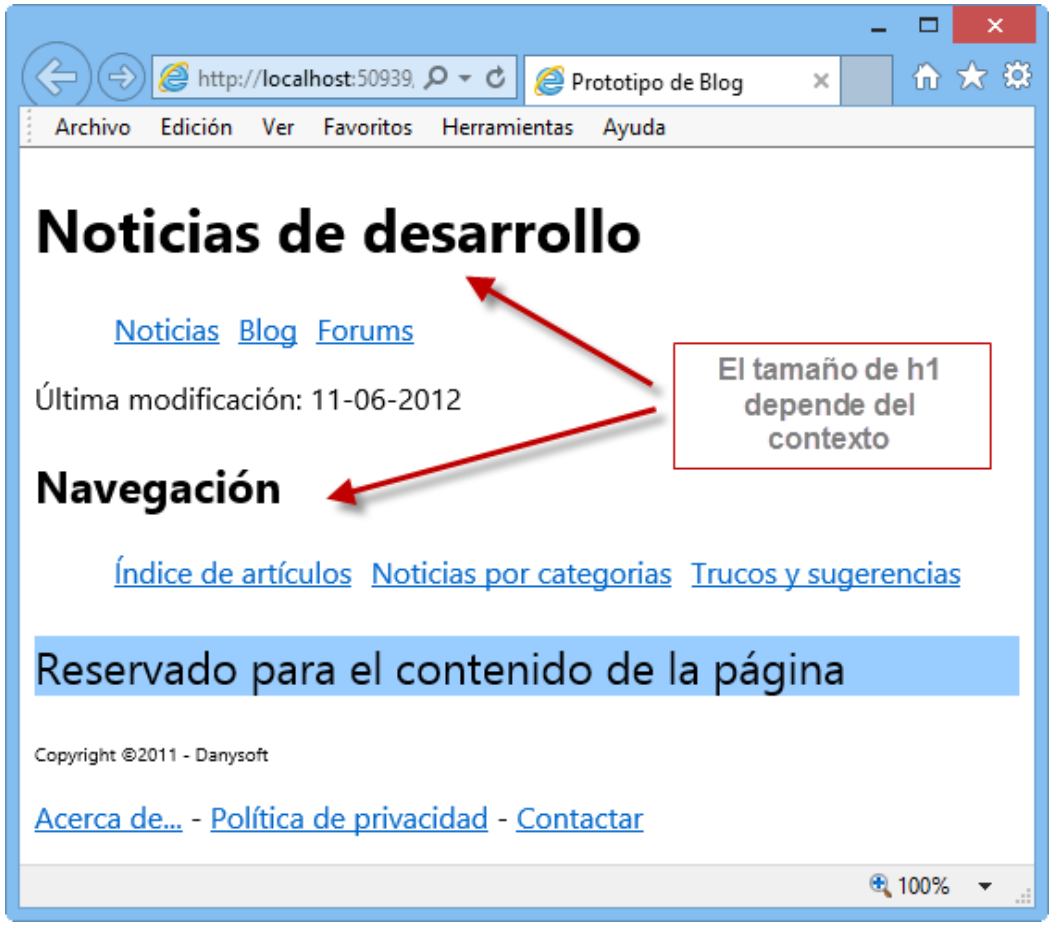


Fig. 5: Salida del código anterior en Internet Explorer

Otro aspecto a tener en cuenta respecto a estas estructuras de navegación tiene que ver con la existencia del elemento **<menu>**, ya

presente en HTML 4.01, pero que dispone de nuevos atributos en esta versión. En realidad, **<menu>** es un elemento pensado para las aplicaciones Web, más que para las páginas. Es fácil que el nombre induzca a error en un principio, pero debemos de tener en cuenta que una buena parte del estándar está ideado con el propósito de potenciar por igual la construcción sitios y aplicaciones Web.

En teoría, un elemento **<menu>** es más apropiado cuando queremos ofrecer una lista de comandos (elementos visuales que apuntan a acciones) dentro de una aplicación Web, y no a un mero cambio de página. La idea de menú sigue ahí, pero se refiere a acciones relacionadas con una aplicación (aunque pueda seguir utilizándose a la manera tradicional).

Lo mismo es aplicable respecto a los menús contextuales, que se supone deben activar acciones que requieren un procesamiento, y no un simple cambio de total o parcial de la información presentada. En el apartado dedicado al elemento **<command>** explicamos con más detalle el funcionamiento conjunto de estos elementos, especialmente para aplicaciones Web.

Implementación de una entrada de un blog

Si llevamos estas ideas a la típica entrada de un blog, en la que el "*blogger*" escribe un artículo y se admiten respuestas y comentarios, podríamos concebir un esquema como el que aparece en la figura 6. Adviértase cómo las respuestas son almacenadas igualmente en elementos **<article>** y se añade un elemento **<nav>**, al objeto de permitir el desplazamiento por los distintos artículos de una sección o por los apartados lógicos del sitio.

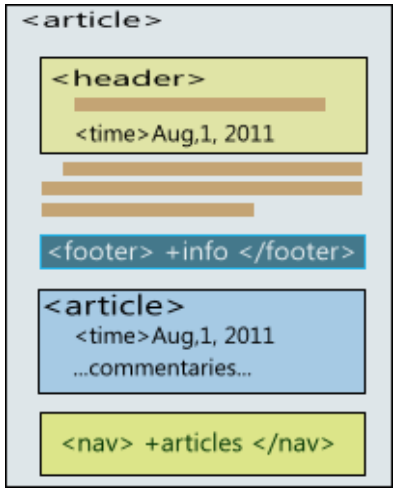


Fig. 6: Estructura de entradas de un blog aplicando la nueva filosofía semántica propuesta por HTML5

A continuación vemos la parte fundamental del código fuente (omitimos los estilos y la cabecera), que podría implementar una idea similar:

```
<section>
    <article>
        <header class="Titular">
            <h3>Lorem ipsum dolor sit amet</h3>
            <time>22 de junio, 2012</time>
        </header>
        <p class="TextoNoticia">
           Sed ut perspiciatis unde omnis iste natus
error sit voluptatem accusantium doloremque laudantium,
totam rem aperiam, eaque ipsa quae ab illo inventore
veritatis et quasi architecto beatae vitae dicta sunt
explicabo. Nemo enim ipsam voluptatem quia voluptas sit
aspernatur aut odit aut fugit, sed quia consequuntur
magni dolores eos qui ratione voluptatem sequi nesciunt.
        </p>
        <footer>
            <em>Imprimatur, Roma.</em>
            <a href="www.Chronicon.ro">(Chronicon)</a>
        </footer>        </article>
```

```
    <h5>Comentarios</h5>
    <article class="Comentario">
        <header>M.Aurelius</header>
        <time>22 junio, 2012</time><br />
        <strong>Et harum quidem rerum facilis est et
expedita distinctio!</strong>
    </article>
        <article class="Comentario">
            <header>Cayo Tulio</header>
            <time>23 junio, 2012</time><br />
            <strong>Temporibus autem quibusdam et aut
officiis debitis aut rerum necessitatibus saepe eveniet
ut et voluptates repudiandae sint et molestiae non
recusandae aut perferendis doloribus asperiores
repellat...)</strong>
        </article>
    </section>
```

Generando una salida como la que vemos en la figura 7:

Fig. 7: Salida en IE9/10/11 del código anterior.

<figure>

Representa un fragmento flotante de contenido, típicamente referenciado como una unidad simple dentro del flujo normal del documento. Habitualmente, se utiliza para albergar elementos gráficos o multimedia a los que aporta metadatos mediante la etiqueta complementaria **<figcaption>** al estilo de lo que muestra el siguiente código fuente:

```
<figure>
        <img src="Graficos/Applications.png" />
        <figcaption>Herramientas</figcaption>
</figure>
```

La salida ubicará la imagen en una pila vertical, debajo de la cual se situará la etiqueta descriptiva. El orden es relevante en el modo de presentación, como era de esperar. Existe un alternativa a **<figcaption>** que se considera obsoleta: **<legend>** que ya no está incluida en el estándar. Ambas etiquetas establecen lo que debe ser una unidad de contenido. Todos los aspectos de presentación son relegados a su formato mediante CSS. Además, el uso de **<figcaption>** favorece la semántica, permitiendo la creación de descripciones más complejas vinculadas con el contenido en sí.

A veces, puede utilizarse para agrupar conjuntos visuales o especiales (respecto al artículo o sección a la que pertenece), que resulta un complemento explicativo adecuado en ese momento: 3 imágenes que muestren 3 vistas de un mismo edificio, con sus descripciones correspondientes, o un vídeo con dos capturas adicionales señalando los puntos a remarcar.

Por supuesto, ese contenido no tiene porqué ser un gráfico (o un elemento multimedia), sino que podría ser cualquier otra cosa, como un fragmento de código fuente.

El siguiente código agrupa 3 tipos de gráficos similares de la biblioteca de imágenes de Windows 7, relacionándolas mediante un elemento **<figcaption>:**

```
<figure>
    <img src="Graficos/Chart.png" />
    <img src="Graficos/Connect.png" />
    <img src="Graficos/Desktop.png" />
    <figcaption>Símbolos habituales</figcaption>
</figure>
```

Lo que (añadiendo un borde a los ***figure***, nos generar una salida como la de la figura 8):

Fig. 8: elementos <figure> y <figcaption> en funcionamiento.

El lector podrá comprobar, además, que el elemento contenedor es de tipo dinámico, por lo que funciona de forma similar a un *WrapPanel*, de forma que, si la superficie disponible no permite albergar los elementos en una sola fila, los desplaza hacia la siguiente, por lo que se ajusta magníficamente a los contextos actuales, donde es preciso programar para dispositivos diversos. En la parte dedicada a CSS, veremos, además, como podemos combinar estas características con las nuevas reglas

dinámicas de Transiciones y Animaciones, y las directivas @**media** que comprueban al superficie disponible en el dispositivo de destino para ofrecer una experiencia de usuario diferenciadora, que se ajuste siempre al contexto de visualización, no requiriendo siquiera del concurso de funciones JavaScript.

Nota sobre este conjunto de etiquetas semánticas

Con estas etiquetas estructurales, estamos añadiendo elementos que relacionan y establecen dependencias entre los diversos fragmentos de código de la página y por tanto, estamos favoreciendo conceptos que tienen que ver con la Web semántica. Estas etiquetas, más que jugar un papel contenedor, establecen relaciones lógicas entre los contenidos.

Eso es algo de lo que no disponíamos hasta ahora. Sus resultados serán a más largo plazo, cuando su uso empiece a generalizarse, los buscadores podrán indexar la información teniendo en cuenta estos criterios, y supuestamente, haciendo que las búsquedas sean mucho más aproximadas –por un lado- y que se "cuelen" menos elementos que no tienen absolutamente ninguna relación con el sentido de lo que buscamos.

Elementos Multimedia: <*video*>, <*audio*>, <*source*> y <*track*>

Una de las novedades más esperadas de esta versión es el soporte nativo de elementos multimedia, lo que –en principio- hace innecesaria la presencia de "*plug-ins*", como Flash y Silverlight, aunque el ámbito de operaciones de éste último es más amplio en la actualidad y se está utilizando más en las aplicaciones de escritorio que en elementos individuales empotrados en una página.

De todas formas, llegados a este punto, nos encontramos con que el apartado de Audio y Vídeo presenta un problema que no existe cuando nos referimos a los gráficos: los códec y los formatos soportados.

Sobre los códec y los formatos soportados

Como no podía ser de otra forma, en el momento de escribir estas líneas, no hay unanimidad en el soporte de formatos. Por un lado, Microsoft y Apple, soportan el formato estándar **H.264** (de extensiones MP4, etc.), Por otro, Google ha creado sus propios códec (**VP8** y, recientemente, **VP9**) y su propio formato (**WebM**, o **WebMedia**) y lo ha propuesto como un formato abierto, pero existían dudas sobre si seguiría soportando el estándar H.264, ya que según Google, "se iba a centrar en formatos abiertos". Afortunadamente, parece que –hasta el momento- Google ha decidido seguir soportándolo.

Y además, nos encontramos con el formato **OGG/Theora**, como formato abierto de audio/video que ya existía con anterioridad, y para el que parece que existen ciertos problemas relacionados con las patentes y licencias de uso, que han hecho que Microsoft y Apple decidan no soportar ese formato (al menos, por ahora).

Y es que la etiqueta <video> se comporta de una forma "agnóstica" respecto al tipo de vídeo que soporta el navegador y la plataforma de ejecución.

Así las cosas, el "estado de la cuestión" (Julio 2013), según la Wikipedia, se recoge en la siguiente tabla:

		Formatos de video soportados			
Navegador	**S. Operativo**	**Theora**	**H.264**	**VP8 (WebM)**	**VP9 (WebM)**
Android	Android	2.3	3.0	2.3	No
Chromium	Todos	r18297	Instalación manual	r47759	r172738
Google Chrome	3.0		3.0	6.0	29.0
Internet Explorer	Windows	Instalación	9.0, 10.0 y 11.0	Instalación	No

Navegador	S. Operativo	Formatos de video soportados			
		Theora	H.264	VP8 (WebM)	VP9 (WebM)
Mozilla Firefox	Windows Phone	manual	9.0	manual	
	Windows RT	No	10.0 y 11.0	No	
Konqueror	Todos	4.4[1]			
Mozilla Firefox	Windows 7+	3.5	21.0	4.0	
	Windows Vista		22.0		
	Linux		24.0		
	Android		17.0		
	Todos		No		
Opera	Todos	10.50	No	10.60	
Safari	iOS	No	3.1	No	
	MacOS X	Instalación manual		Instalación manual	

*Tabla 3: Soporte de formatos de vídeo por parte de los navegadores más populares (fuente: Wikipedia: **http://en.wikipedia.org/wiki/HTML5_video**)*

> **Nota**: Recomendamos que se visite la página en cuestión cuando se tengan dudas sobre el soporte concreto de un formato o un navegador, puesto que se actualiza con regularidad.

Además de los sitios habituales ya mencionados, existen otros sitios especializados en el tema, donde se están publicando herramientas y utilidades para convertir formatos y que disponen de vídeos de código abierto para facilitar las pruebas. Todas las pruebas realizadas en esta obra utilizan las versiones de los navegadores IE9/10/11, Chrome, FireFox y Opera indicadas en esta tabla.

<video>, *<audio>* y etiquetas complementarias

Las dos primeras son auto-descriptivas. Se complementan con los elementos **<source>** y **<track>**, que aportan, respectivamente, el origen del elemento multimedia en más de un formato, e información acerca de las pistas del elemento a reproducir, relacionada con las medidas de tiempo en ese elemento multimedia.

Inicialmente, y bien supuesto que estamos utilizando los formatos y las versiones indicadas en la tabla de más arriba, la puesta en marcha de un vídeo no puede ser más sencilla en su versión básica inicial:

```
<article>
  <figure>
    <video controls="controls" src="big_buck_bunny.mp4" >
    <legend>Video de código abierto en varios formatos.
    </legend>
  </figure>
</article>
```

En este caso, el formato es H.264 (MPEG4), y la salida que presentamos es en el navegador IE10/11, generando (para el artículo en cuestión) la salida esperada. Por defecto, se ha seleccionado la opción **controls** que

fuerza la aparición de la barra inferior de controles para la reproducción.

El vídeo no se inicia automáticamente (salvo que se lo indiquemos ex profeso, mediante el atributo **autoplay**). Como vemos en la figura, disponemos de una interfaz sencilla y en castellano, que permite el avance hacia delante y hacia atrás, indica el tiempo restante, y dispone de opciones de anulación de sonido y expansión a pantalla completa. Naturalmente, el aspecto de esa interfaz varía con cada navegador.

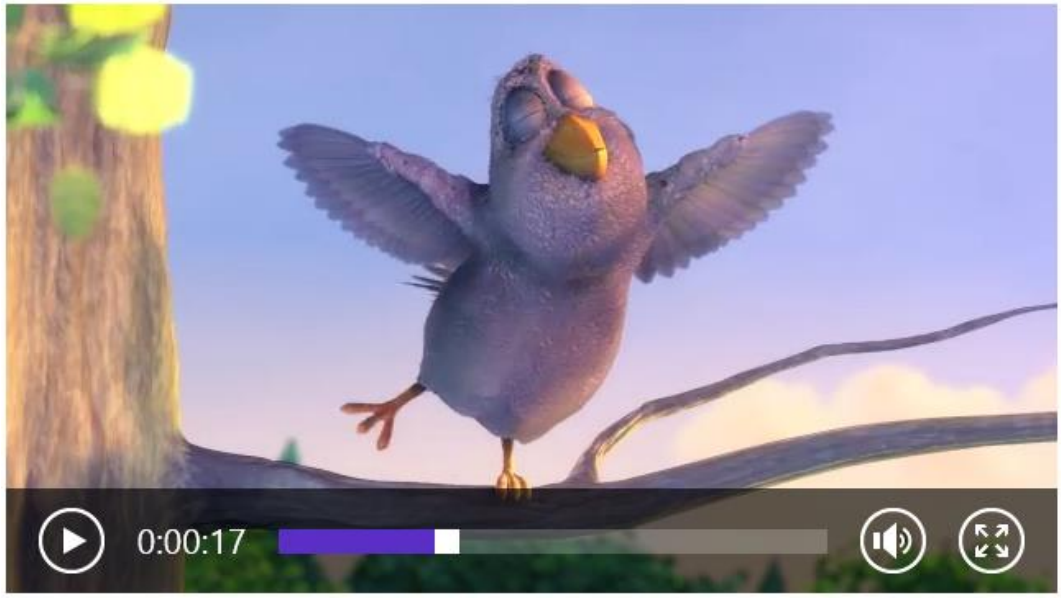


Fig. 9: Vídeo mostrado por IE9/10/11 correspondiendo al código anterior.

Pero, como hemos visto antes, si queremos que nuestro vídeo sea interpretado igualmente por todos los navegadores, o al menos los más populares deberemos de tomar otras medidas adicionales. De hecho, este mismo código no se ve en Opera 16, aunque sí en el resto.

Además del atributo **autoplay** que se usa en el ejemplo siguiente, dispone de una propiedad **poster** que permite indicar una imagen alternativa en caso de que no exista soporte de ese formato de vídeo por parte del navegador. Otras propiedades útiles pueden ser **autobuffer** y **loop**, que, como puede ver el lector, por su nombre resultan bastante auto-descriptivas.

Además, la aceptación de esta etiqueta en la comunidad ha hecho que algunos atributos que tenían carácter de "tentativos" hayan sido realmente bien acogidos y estén soportados por todos los navegadores actuales.

Este es el caso de ***mediagroup*** o ***crossorigin***, para indicar, respectivamente, el tipo de salida multimedia de que se trata o si es posible solicitar una fuente de un dominio distinto del actual.

<source>

Previendo esa posibilidad, el estándar aporta igualmente la etiqueta **<source>** que puede utilizarse para indicar más de un medio en formatos distintos. El navegador del usuario utilizará el primero que considere compatible. Conscientes de esta situación, desde Visual Studio 2012, podemos arrastrar un archivo de vídeo en cualquier formato hacia el editor de código fuente HTML (o hacia el editor visual, si es éste el que se encuentra activo), y el editor nos generará automáticamente el código adecuado para soportar ese vídeo en los 3 formatos. El resultado del proceso, será el que recogemos en el siguiente código fuente:

```
<video controls="controls" autoplay="autoplay">
   <source src="big_buck_bunny.mp4" type="video/mp4" />
   <source src="big_buck_bunny.webm" type="video/webm" />
   <source src="big_buck_bunny.ogv" type="video/ogg" />
</video>
```

Y con esto ya podemos visualizar este fragmento de código correctamente, en cualquiera de los navegadores (si bien, en varios casos, se requerirá la instalación manual del códec, como sucede con Safari que requiere la instalación de QuickTime)[21]. Además, hemos

[21] Para más referencias acerca de la "pesadilla de los formatos", recomendamos visitar las webs siguientes:

- VideoJS: (http://videojs.com) Análisis del soporte del navegador con que se accede y soluciones alternativas para otras versiones

incluido el atributo ***autoplay***, para que el vídeo comience la reproducción inmediatamente después de la carga de la página.

Nota sobre las etiquetas antiguas

Independientemente de esto, HTML5 soporta las etiquetas **<object>** y **<embed>** (y lo seguirá haciendo, en palabras de los responsables de la W3C), que permiten incluir "*plug-ins*" de terceros para la reproducción de vídeo, u otros propósitos tal y como se ha hecho hasta ahora. Esto incluye cualquiera de las versiones habituales de Flash y Silverlight. Más adelante, revisaremos la etiqueta **<embed>**, que ahora está reconocida oficialmente por el estándar.

Respecto a la orientación del vídeo, IE10/11 leen los metadatos del propio vídeo y lo ubica en horizontal o vertical, según su indicación.

<track>

Esta etiqueta permite a los autores especificar explícitamente información sobre las pistas exteriores para los elementos multimedia. No representa nada por sí mismo. El atributo es un atributo de tipo enumerado. La siguiente lista muestra las palabras clave definidas para este atributo.

- **Subtitles**: Traducción de los diálogos, adecuados cuando el sonido está disponible, pero no se entiende (por ejemplo, porque el usuario no entiende el lenguaje de la banda sonora).
- **Legends**: La transcripción del diálogo, propicio cuando la banda sonora no está disponible (por ejemplo, porque se ha silenciado o porque el usuario es sordo).

- HTML5Video.org (http://html5video.org/) Utiliza un mecanismo de "*fallback*" para hacer funcionar el vídeo de HTML 5 en la mayoría de navegadores y versiones.
- The WebM Project (http://www.webmproject.org/): Proyecto de Google sobre su propio formato y códec.

- **Descriptions**: Las descripciones textuales del componente multimedia, destinado a la síntesis de audio cuando el componente visual no está disponible (por ejemplo, porque el usuario está interactuando con la aplicación sin una pantalla mientras se conduce, o porque el usuario es ciego).
- **Chapters**: Títulos de los capítulos, destinados a ser utilizados para navegar por el recurso multimedia.
- **Metadata** : Información sobre las pistas, la orientación, el formato, etc.

Por ejemplo, si disponemos de un fichero de texto con el formato aprobado por el estándar, que disponga de subtítulos para un archivo multimedia dado, podríamos incluirlo en la reproducción mediante una instrucción como la siguiente:

```
<video src="podcast.mp3" controls="controls"
preload="none">
<track src="es_track.vtt" srclang="es" label="Spanish"
kind="subtitles" default="default" />
</video>
```

Donde el archivo ***es_track.vtt***, es un archivo de texto con marcas de tiempo que proporciona comentarios para vídeos o audios. Se pueden usar varias pistas y definir una como predeterminada para usarla cuando comienza el vídeo. El texto se muestra en la parte inferior del reproductor de vídeo. En este momento no se pueden controlar la posición y el color, pero se puede recuperar el texto a través de *script* y mostrarlo como se quiera.

En el caso de que se disponga de más un idioma de sub-titulación el sub-elemento ***<track>*** puede aparecer tantas veces como sea necesario, indicando el idioma, el nombre del archivo y si es el predeterminado, deberá ir marcado con el atributo ***default.***

Al objeto de facilitar la creación de subtítulos, Microsoft ha creado la página "***HTML5 Vídeo Caption Maker***" (disponible en la dirección http://ie.microsoft.com/testdrive/Graphics/CaptionMaker/Default.html),

que permite introducir la URL de un archivo de vídeo, y, mediante un sistema de anotaciones y pausas, ir creando los textos que queremos que aparezcan en la reproducción.

Igualmente, disponemos de la página "***IE10 Video Captioning***" (http://ie.microsoft.com/testdrive/Graphics/VideoCaptions/), donde existe un vídeo explicativo del funcionamiento y creación de los mecanismos de sub-titulación en IE10/11, junto a un ejemplo que explica el funcionamiento.

El formato WebVTT (archivos de extensión .vtt)

Las entradas de texto de estos archivos usan una versión simplificada de los formatos de archivo de texto con marcas de tiempo, llamado "Lenguaje de marcado de texto sincronizado" (***TTML***) o ***Web Video Text Track*** (***WebVTT***). Internet Explorer 10 y las aplicaciones estilo Metro que usan JavaScript actualmente admiten solo indicadores de tiempo y subtítulos de texto.

Los archivos ***WebVTT*** son archivos de texto con formato Unicode de 8 bits (UTF-8) con el siguiente formato:

```
WEBVTT
00:00:01.878 --> 00:00:05.334
Saludos, este es el comienzo del vídeo
00:00:08.608 --> 00:00:15.296
¡Lo primero que vamos a hacer hoy es revisar conceptos
sobre vídeo!
```

La etiqueta WEBVTT se requiere como elemento de comienzo del archivo y debe ir seguida por un avance de línea. Según se nos explica en la documentación oficial: "*Los indicadores de tiempo están en formato HH:MM:SS.sss. Los indicadores Inicio y Fin están separados por un espacio, dos guiones, un signo mayor que (-->), y otro espacio. Los indicadores de tiempo están en una línea con un avance de línea. Inmediatamente después del indicador está el texto de imagen. Los subtítulos de texto pueden tener una o más líneas. La única restricción es que no debe haber líneas en blanco entre líneas de texto. El tipo MIME para archivos WebVTT*

es "text/vtt".

Para más información sobre la construcción de este tipo de archivos, recomendamos visitar la página oficial de Microsoft: http://msdn.microsoft.com/es-ES/library/hh673566.aspx.

Independientemente, se han aportado soluciones en forma de artículos para prácticamente todo tipo de situaciones en las que el vídeo necesite de algún complemento (como los subtítulos). En las referencias del final del capítulo indicamos algunos enlaces a soluciones adicionales para cuando los vídeos están en YouTube.

Además, es posible utilizar opciones avanzadas de tratamiento de vídeo en contextos diversos gracias al API asociada con este elemento., que se discute en el documento oficial[22]. Esto incluye la sincronización de distintos elementos multimedia, selección de pistas concretas de un vídeo en aquellos formatos que lo soporten, etc.

El API Media Elements

Esta API aparece referenciada como "***Media Elements API***" en el documento original y se incluyen casos útiles de –por ejemplo- cómo determinar si podemos usar el elemento ***video*** o un complemento alternativo, cómo poder utilizar un video complementario en lenguaje de signos (por razones de Accesibilidad), etc.

Para ello, el objeto ***MediaController*** dispone muchas funcionalidades que permiten la personalización de la experiencia de usuario en un vídeo según los posibles contextos de ejecución. Esto es extensible igualmente al elemento audio, que vemos a continuación.

El problema de los formatos, no termina ahí, desgraciadamente, y la gran cantidad de códecs existentes (para vídeo y audio), puede suponer que el usuario final no tenga la posibilidad de experimentar la IU que hemos diseñado.

[22] Ver http://www.w3.org/TR/html51/embedded-content-0.html#media-elements

FallBacks

En estos casos, es importante contar con la posibilidad de detectar mediante código que algo va mal y ofrecen la alternativa necesaria para que la experiencia sea lo mejor posible.

Por ejemplo, si un vídeo sólo puede visualizarse mediante unos códec específicos, podemos indicárselo en el código HTML de la siguiente forma:

```
<video controls autoplay>
   <source src="video.mp4" type="video/mp4;
codecs="avc1.42E01E, mp4a.40.2">
   <source src="video.ogv" type="video/ogg;
codecs="theora, vorbis"
        onerror="alternativa(argumento)">
</video>
```

Como vemos, se exigen unos tipos de códec concretos. Lo que podemos hacer en estos casos es suministrar una rutina JavaScript que capture el posible error y ofrezca una alternativa, para lo que hemos asignado el evento ***onerror*** que saltará si se genera un error cualquiera.

<audio>

Como elemento, **<audio>** es en realidad una especificación del estándar que cubre 4 aspectos diferentes, relacionados con el sonido: "***audio input***" (entradas de audio por parte del usuario final), "***playback"*** (reproducción de sonido), ***synthesis*** (síntesis de voz) y "***speech to text***" (capacidad de lectura del texto mediante voz de síntesis), siendo estas dos últimas gestionadas por el mismo grupo de trabajo, y su manejo, -igual que el de "audio input", requiere de programación del API subyacente en JavaScript, por lo que comentaremos algo más sobre el tema en los capítulos finales.

Por el momento, de todos ellos, el más utilizado es, indudablemente, el de reproducción, basado en la etiqueta **<audio>**, que funciona prácticamente igual que **<video>**, disponiendo de los mismos atributos,

y pudiendo utilizarse igualmente en conjunción con las etiquetas **<source>** y **<track>** para aportar definiciones complementarias. Uno de los atributos interesantes, **preload**, permite diferir la carga del elemento multimedia hasta que el usuario requiera la reproducción. Por lo demás, todo lo dicho sobre el funcionamiento de **<video>** es extensible aquí, solo que cambiando una etiqueta por otra, y haciendo referencia a un formato correcto de audio.

Visual Studio 2012/13, también aportan un soporte similar para esta etiqueta, pudiendo arrastrar un elemento de extensión reconocible para el audio sobre el editor de código o sobre la superficie de diseño. El código generado sería similar al siguiente:

```
<audio src="Musica/Adagio%20for%20organ-violin-
strings.mp3" controls="controls" preload="none" />
```

Y el único cambio en el aspecto visual, es –evidentemente- la reducción de la superficie del manejo, que se limita a la zona de controles, como vemos en el gráfico siguiente:

Fig. 10: Reproductor de audio mostrado por IE10/11, como salida del código anterior.

No obstante, y en lo que respecta a los formatos soportados, también existe una discrepancia en cuando al soporte, cuyo estado actual podemos resumir en la siguiente tabla:

Navegador	Formatos soportados por los navegadores				
	Ogg Vorbis	**WAV PCM**	**MP3**	**AAC**	**WebM Vorbis**
Google Chrome	9	Sí	Sí	Sí	Sí
Internet	No	No	9	9	No

Explorer					
Mozilla Firefox	3.5	3.5	21.0, Windows	21.0, Windows	4.0
Opera	10.50	11.00	14	14	10.60
Safari	Sí	3.1	3.1	3.1	No

Tabla 5: Soporte actual de formatos de audio en los navegadores

Las compañías están divididas en cuanto a sus preferencias, y la muchas de las parte de las discrepancias se dan por las distintas licencias de utilización y el miedo a que aparezcan "patentes ocultas", por lo que formatos que inicialmente contaban con la aprobación oficial de W3C, como el formato ***Ogg/Vorbis***, dejaron de aparecer en los borradores, debido a estas causas[23].

Como consecuencia de lo anterior, tendremos que adoptar un sistema de formato múltiple (al igual que con el vídeo), si queremos que el archivo sea reproducible en cualquiera de los navegadores.

<embed>

Técnicamente, este elemento ya estaba soportado por la mayoría de navegadores, pero no formaba parte del estándar, por lo que su aparición no persigue otra cosa que darle cobertura oficial. Dispone de 4 atributos específicos de su función, más los atributos globales y sus eventos correspondientes.

Los atributos fundamentales son:

- **src**: indica la URL del recurso a mostrar
- **type**: el tipo *MIME* correspondiente al contenido
- **witdh** y **height** para establecer los valores de ancho y alto del contenedor.

[23] http://en.wikipedia.org/wiki/Use_of_Ogg_formats_in_HTML5

Un ejemplo de uso de este elemento podría ser el siguiente, que muestra un archivo de extensión ***.swf*** (*Flash*), utilizando este código:

```
<embed src="main.swf" width="550" height="400"
flashvars="id=hello world" wmode="transparent" />
```

Hay que notar que una de las curiosidades de esta etiqueta es la de dejar libertad sobre el soporte de atributos de usuario, a riesgo de implementador del agente de usuario (navegador) correspondiente. El navegador se limita a pasar los parámetros al complemento, que hará el tratamiento necesario.

Por eso el código anterior es válido, permitiendo pasarle al lector de Flash los atributos **flashvars y wmode**, que permiten pasar variables al código de *ActionScript* correspondiente del archivo Flash (*main.swf*, en el ejemplo), e indicarle el tratamiento visual del fondo del complemento, junto un identificador.

Además, disponemos de la etiqueta **<noembed>** para mostrar mensajes en aquellos navegadores alternativos (la opción *fallback*, que siempre debe estar presente) para los agentes que no soporten la ejecución de complementos.

<mark>

Se trata de una etiqueta que permite resaltar parte de un texto, poniendo su fondo en amarillo (de manera predeterminada), como si lo hubiéramos marcado con un rotulador. Admite todos los atributos habituales, incluyendo el atributo *style* que permite modificar el comportamiento visual predeterminado. Por tanto, enfatiza la idea de resaltado en general, dejando totalmente abierta al usuario la definición visual de esta característica.

Por ejemplo si cambiamos el texto de la etiqueta **<legend>** del código de nuestro ejemplo anterior sobre el vídeo rodeando parte de la frase con un elemento **<mark>**:

```
<legend>Vídeo de código abierto <mark>disponible en
varios </mark> formatos</legend>
```

Obtendríamos la siguiente salida por pantalla (Fig. 11)

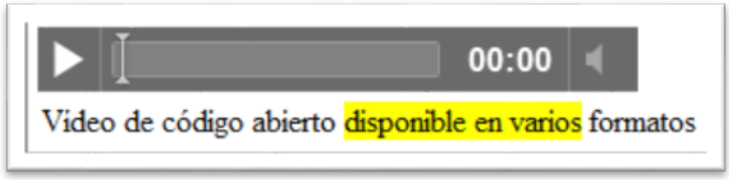


Fig. 11: Salida de la etiqueta <legend> mostrando un texto incluido en etiquetas <mark>

Otro tipo de uso recomendado es cuando queremos resaltar las partes de un documento o aplicación de las cuales el usuario esté realizando una búsqueda. Igualmente, el estándar remarca la diferencia entre "importancia" de una frase (que podríamos marcar con un elemento ***strong***) y su "relevancia", que tiene que ver con el sentido de la tarea que se realiza.

Podemos ver esta diferencia en la adaptación siguiente del ejemplo oficial:

```
<h3>Wormhole Physics Introduction</h3>
    <p>
       <mark>
        Momentum is preserved across the wormhole.
        Electromagnetic radiation can travel in both
        directions through a wormhole, but matter cannot.
       </mark>
    </p>
    <p>
     When a wormhole is created, a vortex normally forms.
    <strong>
     Warning: The vortex caused by the wormhole opening
     will annihilate anything in its path.
        </strong> Vortexes can be avoided when
        using sufficiently advanced dialing technology.
```

```
    </p>
    <p>
       <mark>
        An obstruction in a gate will prevent it from
        accepting a wormhole connection.
       </mark>
    </p>
```

La salida de este código muestra estos dos aspectos del código resaltado: la importancia y la relevancia, como muestra la figura adjunta:

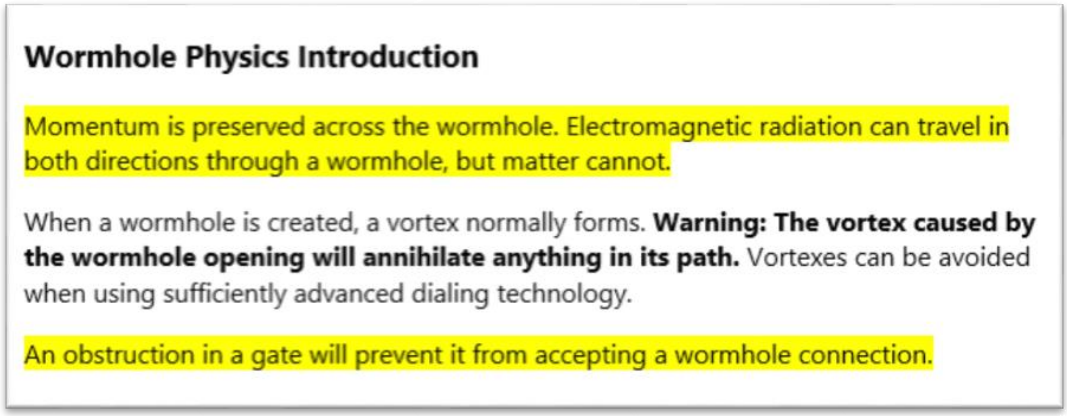


Fig. 12: Salida del código oficial del estándar enfatizando la diferencia de los dos conceptos

<progress>

Indica el grado de finalización de una tarea y es especialmente útil en procesos de larga duración como una descarga de archivos o la ejecución de cálculos pesados. Está pensado para ser usado junto a JavaScript en un proceso dinámico que va indicando los valores porcentuales de realización de la tarea.

En su implementación final, el usuario podrá diseñar un mecanismo visual de progreso y empotrarlo dentro de dos etiquetas de este tipo, haciendo que un evento programado se encargue de la actualización de sus valores.

Admite los atributos **Max** y **Value**, para indicar, respectivamente, los valores actual y máximo del proceso (los valores predeterminados son 0 y 100). El nivel de soporte en el momento de escribir estas líneas es diverso. Todos los navegadores que analizamos muestran el contenido, pero en algún caso, al detectar esta etiqueta se presenta una salida gráfica inspirada en las interfaces similares de aplicaciones de escritorio (como el elemento *ProgressBar*, en el caso de .NET y Windows).

El siguiente código ilustra una situación típica:

```
<section>
    <h2>Indicadores de Progreso</h2>
    <p>Progreso:
        <progress id="progreso" max="100"
value="0"></progress> 
        <span id="progresoNum">0</span>%
    </p>
</section>
<script>
    var progressBar =
document.getElementById('progreso');
    var contador = 0;
    var temporizador =
setInterval("actualizar(contador)", 1000);

    function actualizar(valor) {
        progressBar.value = valor;
        document.getElementById('progresoNum').innerHTML
= contador;
        if (contador >= 100) {
            clearInterval(temporizador);
            return;
        }
        contador += 20;
    }
</script>
```

Como vemos, se define una variable de recuento (***contador***) que vaya incrementando el porcentaje y una función que será llamada por un

temporizador, inicializado mediante ***setInterval***, y asignado a una variable, que será pasada al mecanismo de parada condicional cuando se llega al 100.

Cada segundo, la función va incrementando el valor y actualizando la interfaz de usuario, que es doble: el elemento ***<progress>***, y una sencilla etiqueta ****** utilizada como mecanismo de "fallback", para los casos en los que todavía el navegador no ha implementado una interfaz visual para el elemento.

La figura 13 muestra la salida en el navegador Opera:

Fig. 13: Salida del código anterior en Opera.

<meter>

Representa una medición, o un valor escalar pero dentro de un rango, como puede ser el porcentaje de uso de un disco duro. No tiene el carácter dinámico de la anterior, aunque sus valores puedan variar con el tiempo. Pero tampoco está pensado para presentar información escalar estándar, sino solamente aquella que se muestra en referencia a un valor máximo (valores porcentuales, o que implican ese concepto).

Dispone de los atributos (fundamentales): **value**, **min**, **max**, **low**, **high** y **optimum**. Los 3 primeros resultarán bastante evidentes (similares a los anteriores). Los otros 3 son complementarios, pudiendo expresar un valor considerado como bajo por el autor (**low**), otro considerado como alto (**high**) y uno considerado como óptimo (**optimum**).

El nivel de soporte por los navegadores es bastante primario, a excepción de Chrome, que muestra la información –en su forma básica- como una barra (similar a la de la etiqueta **<progress>**), pero sin la parte dinámica

de aquella. En el resto de navegadores, sucede algo similar a lo anterior. Sólo se muestra el valor textual, pero sin ninguna interpretación gráfica.

Veamos algunos ejemplos de uso con sus salidas, especialmente en Chrome, que ya ofrece soporte visual de los datos.

```
<p>Ha completado: <meter value="5">5 de un total de 10
</meter> </p>
<p>Sus Respuestas: <meter min="0" max="10" value="3">3 de
un total de 10</meter></p>
<p>Evaluación del examen: <meter value="67" min="0"
max="100" low="40" high="80" optimum="100"> B</meter></p>
```

Como vemos en el código anterior, la primera línea utiliza la versión más simple, indicando exclusivamente los atributos *value* y *max*. Esta opción no se considera correcta por el estándar, ya que este elemento expresa el grado de compleción de una magnitud *dentro de un rango*. En el segundo, se indican correctamente los 3 valores base, y en el tercero, especificamos los otros 3 parámetros, lo que produce un cambio en el color de relleno de la barra, indicando un valor próximo al óptimo, pero no mucho.

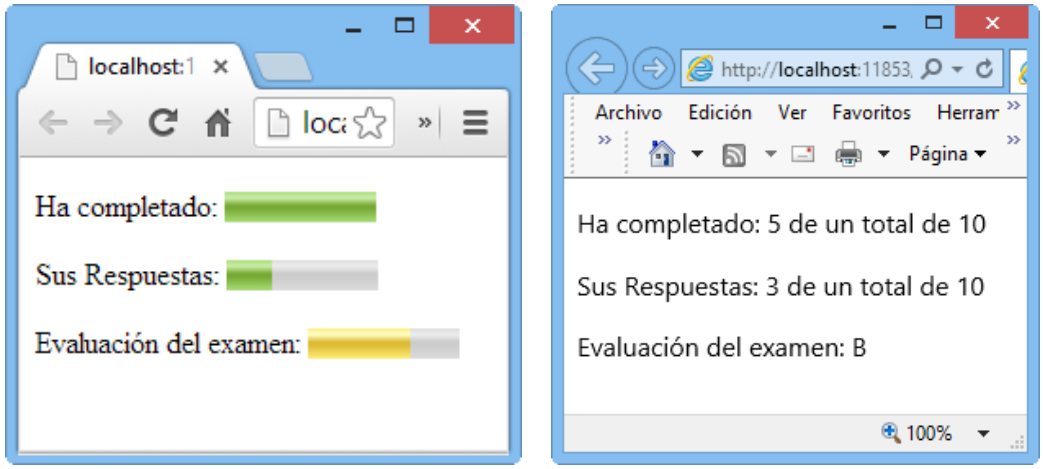


Fig. 14: Salida del código anterior en Chrome e IE11, respectivamente.

De momento, IE no ofrece soporte visual de este elemento (aunque habrá

que esperar a la aparición de la versión final de IE11).

<time>

Es uno de los elementos a los que el estándar se refiere como micro-sintaxis, y que anteriormente se han identificado bajo el nombre de Micro-formatos.

En sí, la etiqueta representa una fecha y/o una hora. Puede representar una fecha en calendario gregoriano, o bien una hora indicada en un reloj de 24 horas, incluyendo información de zonas horarias. El formato reconocible de cadenas de fecha y hora válido para el estándar también está especificado con detalle en el documento oficial del estándar (Aptdo. 2.5.5 *Dates and Times*).

Su función es principalmente semántica, ayudando en los procesos de conversión y/o comprensión de diversos formatos de fecha y hora en contextos multiculturales. Por ello, permite indicar la fecha/hora en el atributo ***datetime*** mientras que el cuerpo de la etiqueta puede contener una descripción más adecuada al contexto, o no tan formal, o expresada en un lenguaje alternativo.

Esta forma de expresar fechas y horas, también está claramente indicada en el estándar como una de la llamada "*micro-syntaxes*", y podemos ver su descripción detallada el documento oficial[24].

En su forma más simple (sin atributos) puede escribirse de esta forma:

```
<time>2011-10-15</time> //indica una fecha
```

Para este caso, queda entendido que el texto entre las etiquetas debe ser una fecha o una hora válida de acuerdo con los formatos de cadena indicados en la especificación.

No obstante, si utilizamos el atributo para la indicación del valor,

[24] http://www.w3.org/TR/2012/WD-html5-author-20120329/common-microsyntaxes.html#dates-and-times

podemos optar por otras opciones, tal y como vemos en el siguiente código.

```
<!-- Si está presente el atributo @datetime -->
<p><time datetime="2011-10-15">24 de Diciembre, 2011
</time></p>
<!-- time -->
<p><time datetime="16:00">Apertura a las 4:00
pm</time></p>
<!-- datetime incluyendo time-zone -->
<p><time datetime="2011-10-15T16:00+00:00">4:00 pm
próximo sábado</time></p>
<!-- datetime incluyendo time-zone formato "Z"-->
<p><time datetime="2011-10-15T16:00Z">4:00 pm próximo
sábado</time></p>
```

No debemos confundir esta etiqueta con la etiqueta ***<input type="time">*** que está pensada para ofrecer alternativas sencillas al usuario para introducir este tipo de información en un formulario, por ejemplo. Trataremos las variantes de esta etiqueta en el siguiente capítulo dedicado a los nuevos atributos de elementos prexistentes.

La salida generada es no ofrece ninguna sorpresa visual, ni interpretación alternativa, pero los agentes de usuario pueden interpretar la fecha en la forma que mejor les convenga, sin depender del contenido de la etiqueta, más que como una mera descripción adicional:

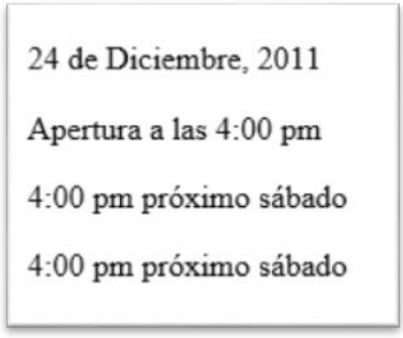


Fig. 15: Salida del código anterior de etiquetas <time>

La especificación indica otros atributos posibles, además de los globales y de eventos, como **pubdate** (indicar una fecha de publicación) y **title**, que ayudarían a comprender mejor la información asociada al elemento.

<rt> y <rp>

Permiten marcar anotaciones ***Ruby*** en tipografías del Este de Asia. Se complementa con las otras dos etiquetas **<rt>** y **<rp>**. Consta de uno o más caracteres expresados en una de esas lenguas, y se usan para explicar la pronunciación vinculada con el texto. La explicación se incluye en elementos **<rt>**, y puede aportar una descripción alternativa en la etiqueta **<rp>** para los navegadores que no soporten el elemento **<ruby>**.

<bdi>

Representa un intervalo de texto bidireccional que debe ser aislado de su entorno a efectos de formato de texto. Evita los errores que se producían en ocasiones debido al uso que el Algoritmo Bidireccional de Unicode (*Unicode Bidirectional Algorithm*[25]) hace de los llamados "*weak characters*". Por ejemplo, si queremos mostrar un texto de derecha a izquierda, y utilizamos un elemento **** junto al atributo **dir="rtl"** (abreviatura de "*right-to-left*"), nos podemos encontrar con que la salida se ve afectada por esos caracteres.

Veamos el ejemplo siguiente (tomado del sitio oficial de la W3C):

```
<ul>
    <li>User
       <bdi>jcranmer</bdi>: 12 posts.</li>
        <li>User <bdi>hober</bdi>: 5 posts.</li>
        <li>User <bdi>إيان</bdi>: 3 posts.</li>
</ul>
```

La línea del tercer usuario aparecería incorrectamente utilizando la sintaxis ****, mientras que de esta forma, se debe generar

[25] Ver el sitio de Unicode http://unicode.org/reports/tr9/ para la explicación "oficial".

una salida correcta.

El soporte de navegadores es completo. En navegadores antiguos, simplemente ignora todo lo que sigue, con lo que se pierde la parte derecha del contenido. La salida correcta (la actualizada) es la que vemos en la figura 16.

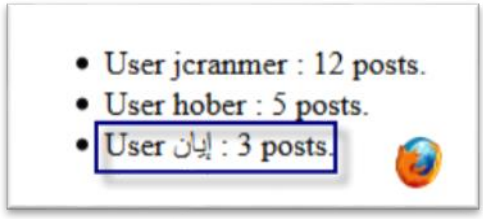


Fig. 16: Salida del código anterior en Firefox (es idéntica en los otros navegadores).

<wbr>

Cuando tenemos que representar una cadena muy larga sin espacios, como puede ser una URL, nos permite establecer un lugar donde puede situarse un salto de línea. Esto evita problemas de formato, o que se produzcan divisiones de la cadena en zonas que resulten de más difícil comprensión.

Por ejemplo, cuando el contexto de presentación se reduzca, (porque el usuario reduzca el tamaño de la ventana, o se visualice en una pantalla de pequeñas dimensiones) el siguiente código ofrece 3 oportunidades de división para la palabra larga del final, favoreciendo la lectura:

```
<p>Julie Andrews, en "Mary Poppins" cantaba una canción
llamada
“Super<wbr/>cali<wbr/>fragilistico<wbr/>expialidoso".
</p>
```

El soporte es completo en la mayoría de navegadores (Excepto Internet Explorer, a falta de comprobar la versión final de IE11).

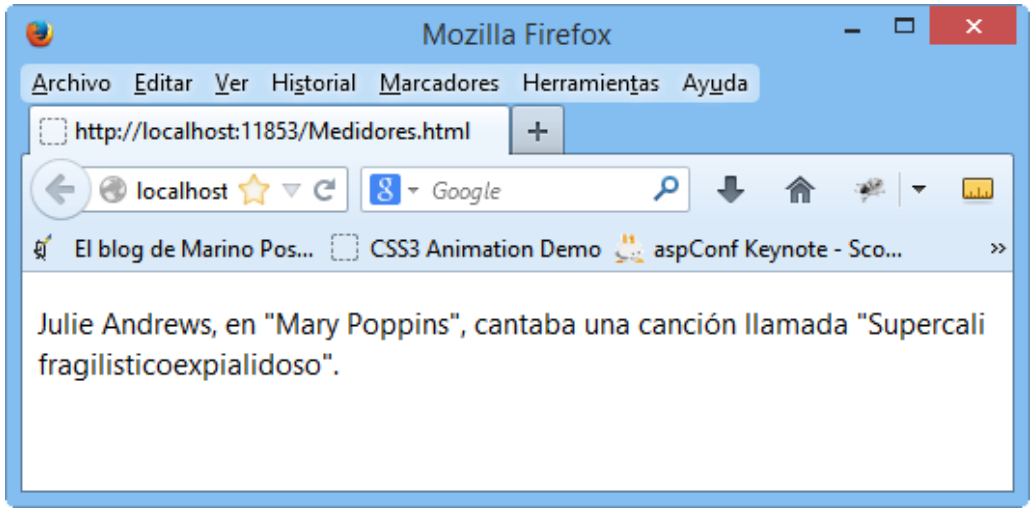


Fig. 17: El elemento <wbr> en funcionamiento en FireFox

<command>

Aunque apareció desde los primeros borradores de la especificación, no ha contado con el beneplácito de los creadores agentes de usuario, por lo que se encuentra, según indicábamos al principio, en "riesgo".

Ya apuntábamos en el elemento **<menu>** que buena parte de las etiquetas están pensadas para las aplicaciones Web, y no sólo para las páginas. Ese es el caso de **<command>**.

La idea es que se use en combinación con **<menu>** (u otros) para programar acciones genéricas que pueden ser invocadas desde distintos puntos de la aplicación: *Añadir*, *Borrar*, *Cortar*, *Pegar*, *Abrir Fichero*, etc., de forma que solo tengamos que definir el comando y podamos invocarlo desde diversos lugares, sin repetir la parte funcional.

Por ello, un elemento **<menu>** dispondrá de un atributo **<command>**, que apuntará un elemento de este tipo. A su vez, éstos son configurables mediante el atributo ***type***. Y lo mismo puede decirse de los elementos ***<input>***, que ahora poseen un atributo ***command.***

Para el caso de ***<menú>***, este atributo puede adoptar 3 valores distintos:

- **command** (el identificador del comando a invocar)
- **radio** (usado en combinación con una marca **radiogroup**)
- **checkbox**

Un caso práctico simple, podemos verlo en el código siguiente:

```
<command type="command" id="cmd" onclick="Accion()" />
<input type="button" command="cmd" value="Llamar al
comando" />
<menu command="cmd" >Llamar al comando desde Menú</menu>
<script>
    function Accion() {
       alert("Comando invocado");
    }
</script>
```

En el ejemplo, tanto si pulsamos en el botón (elemento ***<input>***), como si lo hacemos en el elemento ***<menú>***, se producirá la llamada a la función "**Accion**()", y lanzará la ejecución del código JavaScript correspondiente, como podemos ver en la figura 14, aunque –estrictamente hablando- solo lo hemos codificado una vez (en el ***command***). Todos los elementos que soportan el atributo *command* asumen este comportamiento.

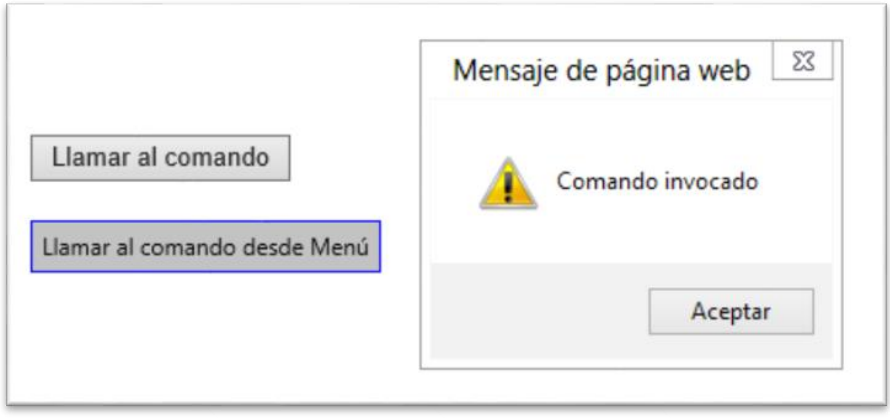


Fig. 18: Ejecución del código anterior, desde Internet Explorer 9/10

La salida muestra el funcionamiento en Internet Explorer 9/10. Por el momento, esas versiones de IE son las únicas que soportan esta característica, por lo que podría quedar fuera del estándar si esa situación continúa.

<keygen>

Representa un mecanismo de control para la generación de un par de datos (clave pública/clave privada) o *key/value pair* de tipo criptográfico. Cuando el formulario al que pertenece el control es enviado, se almacena de forma local la clave privada, y se envía la clave pública.

Además de los habituales, posee los atributos específicos ***challenge*** y ***keytype***, que permiten establecer respectivamente un valor original de partida e indicar el tipo de algoritmo de cifrado que se desea utilizar. El siguiente código genera una salida como la de la figura 14 (respectivamente, para Chrome, Firefox e IE9/10/11):

```
<form method="post" action="Generar.aspx">
        <keygen name="RSA PK" challenge="9827492834"
keytype="RSA" />
        <input type="submit" name="gencert"
value="Generar" />
</form>
```

Lo que genera la misma maquinaria operativa, pero distintas presentaciones, dependiendo de que se trate de IE9/10/11, FireFox o Chrome. En todos los casos, un botón "*Generate Key*", cuando es pulsado genera el par de claves e invoca la página asignada al atributo ***Action***.

Curiosamente, en las versiones de FireFox, Chrome y Opera, se añade un *ComboBox* automáticamente donde es posible la selección del nivel de cifrado (medio/alto).

Fig. 19: Salida del código de generación del par de claves en Chrome, FireFox e IE10/11.

(Recuerde que debe de existir una página de nombre "Generar.aspx" en el servidor, o al pulsar el botón de envío, recibiremos un mensaje de error).

> **Nota Importante:** Existen serias dudas sobre el soporte final de esta etiqueta en las versiones finales de los navegadores debido a la forma en que interactúa con el DOM, afecta considerablemente a los modelos de envío y validación de formularios, y, además, la mayor parte de los sitios que utilizan esta tecnología están usando otros mecanismos bien probados, dada la importancia fundamental en el comercio, especialmente.

En el caso de Microsoft, parece que la tecnología equivalente que está siendo usada se basa en el API de Certificados de Inscripción (*Certificate Enrollment API*)[26].

Por si esto fuera poco, tanto Apple[27], como Google han manifestado su no intención de soportar esta etiqueta. Google la implementó en

[26] http://msdn.microsoft.com/en-us/library/windows/desktop/aa374863%28v=vs.85%29.aspx

[27] http://images.apple.com/iphone/business/docs/iPhone_OTA_Enrollment_Configuration.pdf

Android, pero existen serias dudas sobre su continuidad.

<output>

Indica algún tipo de salida, como la que tiene lugar después de un cálculo mediante *scripting*. Si el elemento ***<input>*** ha visto mejoradas sus posibilidades considerablemente (lo veremos con más detalle en el siguiente capítulo), ahora disponemos de un elemento ***<output>***, que nos podría permitir almacenar una fórmula para un cálculo y mostrar su resultado de forma automática sin necesidad de llamar a ningún *script*.

Para ello, programamos una definición asociada al formulario que contiene los elementos de entrada y salida, que –en este caso- realiza una simple operación de suma, programando la formula en los eventos del formulario contenedor:

```
<form onsubmit="return false"
oninput="resultado.value = n1.valueAsNumber +
n2.valueAsNumber">
<input name="n1" type="number" /><span>+</span>
<input name="n2" type="number" /><span>=</span>
<output id="resultado" for="n1 n2">
</output>
</form>
```

Observe que no necesitamos enviar nada al servidor, por lo que el evento ***onsubmit*** es asignado a "***return false***". Cuando se produce una entrada el evento ***oninput*** asigna dinámicamente el resultado a la propiedad ***value*** del elemento ***output***

El soporte parece que va a ser completo, por parte de todos los navegadores, aunque, todavía no se ha conseguido el soporte completo (por ejemplo, Firefox e IE no soportan aún el atributo ***valueAsNumber***).

La salida del código anterior en los navegadores, adopta una forma similar al de la figura 20:

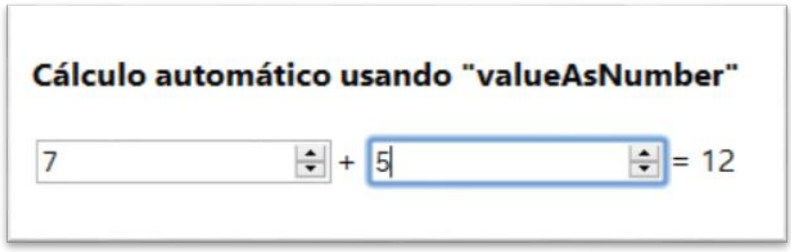


Fig. 20: Salida del código anterior en Chrome/Opera

Si queremos implementar esta funcionalidad ahora mismo y obtener soporte del resto de navegadores, disponemos de un mecanismo de *fallback* muy sencillo, consistente en sustituir la referencia al elemento ***output*** por el correspondiente método JavaScript (***getElementById***), y podríamos reformular este código de la siguiente forma:

```
<form onsubmit="return false"
        oninput="document.getElementById('o').innerHTML =
        parseFloat(document.getElementById('a').value) +
        parseFloat(document.getElementById('b').value)">
        <input id="a" type="number" step="any" /> +
        <input id="b" type="number" step="any" /> =
        <output id="o" for="a b"></output>
</form>
```

Hecho esto, la implementación es correcta, y obtendríamos la siguiente salida en IE10/11:

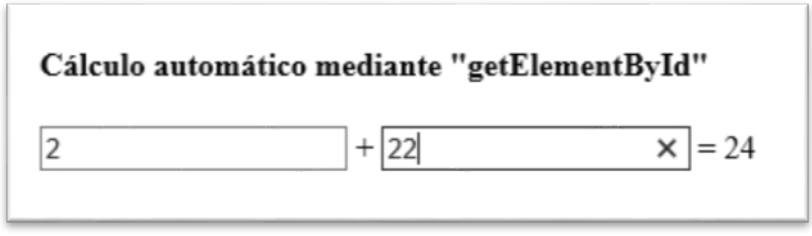


Fig. 21: Salida en Chrome/Opera del código anterior en IE10/11 y Firefox

Se trata de otro de los elementos que el estándar califica como "en estado de riesgo", debido precisamente a la falta de soporte (Solo el motor de Chrome que, ahora, se extiende también a Opera).

<details> y <summary>

Simula una situación similar a la del control *expander* en aplicaciones de escritorio. Permite mostrar u ocultar un conjunto de ítems, ofreciendo al usuario un sencillo mecanismo de selección mediante el cual se muestra una ventana plegable con el resto de los datos adicionales (los detalles, en realidad).

Podemos complementar con el elemento **<summary>** para ofrecer su resumen, leyenda o título.

```
<details>
   <summary>Quién es Danysoft</summary>
   <img src="Graficos/LogoDanysoft.png" />
   <p>Danysoft es un grupo español, que desde 1990...</p>
</details>
```

Esto genera una salida en Chrome/Opera (el único que lo soporta por el momento), como la de la figura 22:

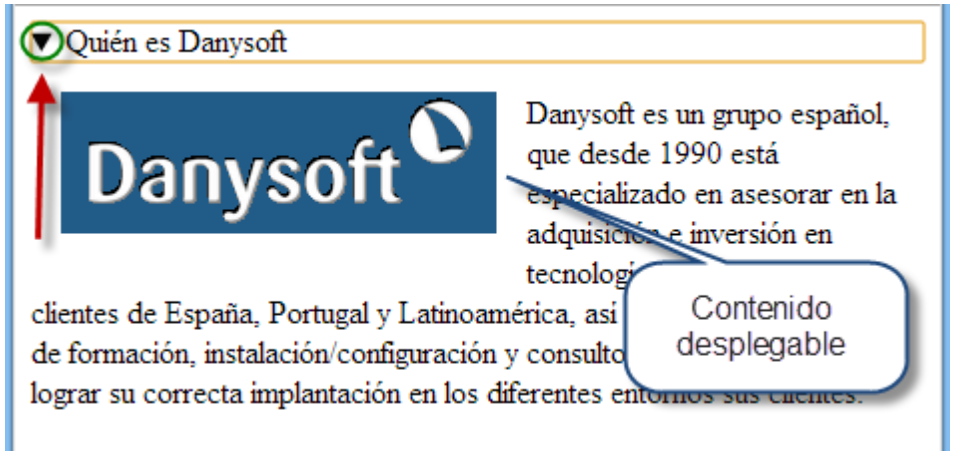


Fig. 22: Aspecto del elemento <details> una vez expandido (la flecha apunta abajo)

La propiedad ***open*** (atributo de presencia), permite que se establezca cómo deseamos que el elemento aparezca en su visualización inicial, siendo cerrado la opción predeterminada.

Al igual que ***output*** se trata de otra etiqueta calificada como "de riesgo" por la falta de implementación.

<datalist>

Junto con el nuevo atributo ***list*** para el elemento ***input***, puede utilizarse para hacer cuadros combinados (***ComboBoxes***), que contengan un conjunto de elementos inicial.

Junto con las nuevas opciones del elemento **<input>**, se relaciona con la creación de los nuevos modelos de formularios que soporta HTML 5, (nuevos aspectos visuales para controles, nuevas capacidades para los ya conocidos y mecanismos de validación de entradas, entre otras características).

De cualquier forma, por completar la descripción, valga decir que una adaptación del ejemplo estándar[28] para esta etiqueta, sería la siguiente:

```
<span>Seleccione una ciudad</span>
<input type="text" id="city" list="ciudades">
<datalist id="ciudades">
    <option value="Madrid">
    <option value="Barcelona">
    <option value="Valencia">
    <option value="Sevilla">
</datalist>
```

Y, en tiempo de ejecución, obtendríamos la siguiente salida en todos los navegadores (el soporte es completo en las últimas versiones):

[28] http://www.w3.org/wiki/HTML/Elements/datalist

Fig. 23: Salida del código del elemento <datalist> en IE10

A continuación dedicamos la parte final de este capítulo a dos elementos exclusivamente dedicados a dar soporte a la construcción de gráficos dinámicos: del tipo "mapa de bits" en un caso (***canvas***) y de carácter vectorial en el otro (SVG), recordando que lo nuevo de este último no es su existencia, sino su soporte en los navegadores, ya que se trata de una especificación existente desde varios años atrás, pero que no era muy utilizada. En HTML 5, **SVG** complementa la oferta gráfica con un sistema vectorial de calidad.

Gráficos: Mapas de bits (<*canvas*>) y vectoriales (<*SVG*>)

Hemos visto cómo el estándar aporta alternativas interesantes para poder prescindir de los complementos que hemos usado hasta ahora en lo referente a la multimedia (principalmente, Flash y Silverlight). Pero el propósito de la nueva Web estaría incompleto sin mecanismos que permitan sustituir a esos complementos en lo que referente a los gráficos, efectos visuales, animaciones, transformaciones, etc. Por ello, se ha optado por un doble enfoque, en función del tipo de resultado deseado, el contexto de ejecución, soporte, etc.

Por un lado, existía la necesidad de un mecanismo que aportara velocidad en el procesamiento (fundamental en muchas aplicaciones, pero absolutamente imprescindible en los juegos, por ejemplo), además de la posibilidad de incorporar y manipulación de gráficos del tipo "mapa de bits" y realizar todas las operaciones a que los complementos nos

tienen acostumbrados y más.

Para ello, se creó el nuevo elemento ***<canvas>***, que no es un elemento más, sino un API completo de dibujo no vectorial, pero de excelente rendimiento. Es fácil de implementar, y posee un conjunto de palabras reservadas para definir las acciones a realizar, mediante un lenguaje propio, de sintaxis tipo JavaScript.

Por otro lado existen situaciones en las que resulta imprescindible la utilización de gráficos de tipo vectorial, que permiten el escalado sin pérdidas, transformaciones avanzadas de calidad, y un montón de operaciones gráficas similares a las que podemos hacer con *ActionScript* para **Adobe Flash** o con un lenguaje .NET Framework para **Silverlight**. En este caso, no era preciso crear un nuevo API, sino solamente adaptar el ya existente (***SVG***), por lo que la W3C publicó el verano de 2011 su "*Candidate Recommendation*" (versión 1.1) del estándar SVG, con la petición urgente de que los agentes de usuario se apresuraran a soportarlo lo antes posible, cosa que podemos afirmar que se ha cumplido en todos los navegadores principales que analizamos aquí (el lector dispone en el apartado de referencias de este capítulo de enlaces de internet y bibliografía para poder profundizar en el estudio de estos elementos).

<canvas>

Es uno de los que más "revuelo" ha generado entre las novedades esperadas de HTML 5, y merece un tratamiento especial. Como hemos apuntado, su objetivo es permitir el procesamiento dinámico de mapa de bits gráficos o juegos, aunque también dispone de herramientas simples de dibujo.

Pros y contras

Dado que hemos hablado antes de SVG, como mecanismo de dibujo vectorial, conviene –antes de nada- ver cuáles son las diferencias con este sistema de dibujo. Como dice su definición, se trata de un sistema de trazado que genera mapas de bits (*bitmaps*), por lo que no soporta las

opciones de escalado, deformación, etc., típicos de los gráficos vectoriales, y –lo que es peor- no forma parte del DOM (Modelo de objetos del documento) o de ningún otro espacio de nombres. Además, tampoco es capaz de detectar interacciones con el usuario (pulsaciones en una zona de su superficie, por ejemplo).

La parte positiva: el rendimiento es bueno, debido a que no tiene que almacenar definiciones de primitivas de dibujo. El cambio estriba en que no tienen que descargarse los gráficos del servidor, y los motores de JavaScript actuales aportan un gran rendimiento en el proceso. Esto es totalmente coherente con la arquitectura de las aplicaciones AJAX, que tiende a procesar en el cliente la mayor cantidad de elementos.

Además, resulta sencillo de implementar una vez que conocemos alguna de las API bidimensionales existentes en otros lenguajes de programación, lo que significa que las instrucciones y nomenclatura de los elementos que se usan para el dibujo, probablemente le resultarán familiares al lector que haya implementado algún mecanismo similar en los últimos años.

Recomendaciones previas

La propia especificación indica que no debe usarse **<canvas>** cuando existan otros elementos de la especificación más adecuados para ello. Si es posible realizar lo mismo utilizando elementos HTML y CSS es preferible optar por esta posibilidad.

El otro problema que podemos encontrarnos es que el navegador del usuario no soporte el elemento **<canvas>**, por lo que conviene establecer un mecanismo alternativo (el *fallback* de marras*)* que recuerde al usuario esa circunstancia, ya sea mediante un texto explicativo o una imagen sustitutiva, como vemos en:

```
<canvas id="CanvasID">
<img src="canvasElement.png" alt=" Su navegador no
soporta CANVAS" />
</canvas>
```

¿Existe alguna alternativa en caso de que el navegador del usuario no soporte el elemento **<canvas>**? En principio, no se producen errores, simplemente no se muestra el contenido. Pero, si tenemos que mostrar un contenido realizado con el API de ***canvas***, podemos recurrir a una librería llamada *ExplorerCanvas*[29], en el caso de que se trate de IE6/7 u 8. En IE9 el soporte es muy completo, y en IE10/11, gracias al nuevo motor de JavaScript y al aprovechamiento de las capacidades del hardware de la máquina cliente, el rendimiento es espectacular.

Una vez descargada la librería, podemos hacer referencia a ella con una sencilla línea de script, como vemos a continuación:

```
<!--[if lte IE 8]>
  <script src="excanvas.compiled.js"></script>
<![endif]-->
```

Soporte en Visual Studio 2012/2013

El soporte para trabajar con ambas API de dibujo en Visual Studio 2012 es total. No solo en la parte del editor (*Intellisense*, enumeración de miembros, corrección sintáctica, etc.), sino en otros aspectos como la interpretación visual, el análisis del código vinculado con las instrucciones de dibujo, depuración, detección de características, etc. Lo iremos viendo según avanzamos en este repaso por el elemento.

Programación general de un objeto *canvas*

Cuando dibujamos en un objeto **canvas**[30], lo primero es obtener una referencia al propio elemento a través de cualquier mecanismo de JavaScript, como, por ejemplo, una llamada a ***getElementById*()**. Una vez que tenemos esa referencia podemos establecer un contexto de trabajo

[29] Disponible en http://code.google.com/p/explorercanvas/

[30] El documento oficial de la especificación de <*canvas*> en la W3C se encuentra en la dirección http://dev.w3.org/html5/2dcontext/.

(objeto ***context***), que será el encargado de aportar los métodos y propiedades necesarios en programación.

Un sencillo *script* nos permite determinar la presencia o ausencia de soporte en el navegador mediante el siguiente código:

```
<p id="SoporteCanvas">Soporte de canvas en su
navegador:</p>
<script>
    try {

document.createElement("canvas").getContext("2d");

document.getElementById("SoporteCanvas").innerHTML +=
"soportado";
    } catch (e) {

document.getElementById("SoporteCanvas").innerHTML += "no
soportado";
    }
</script>
```

Con esto como inicio del proceso, ya podemos continuar, sabiendo que todos los navegadores con los que trabajamos tienen actualmente soporte completo de este elemento.

Accediendo a *canvas* para dibujar

En la primera línea observamos el siguiente paso a seguir, una vez que disponemos de una referencia a **canvas**: obtener una referencia a su contexto, o sea, a su superficie de trabajo, para ir indicando operaciones a realizar sobre ella.

Esa superficie de trabajo viene delimitada por los atributos que nos permiten delimitar la superficie de trabajo (aparte de los genéricos): anchura (***width***) y altura (***height***). Los valores respectivos a las coordenadas se asignan, como era de esperar comenzando por la esquina superior izquierda, en la forma habitual en computación. Si bien, cuando contamos a partir del origen se asume que los valores de la x

desplazan hacia la derecha y los valores de la y hacia abajo (Ver Fig. 24):

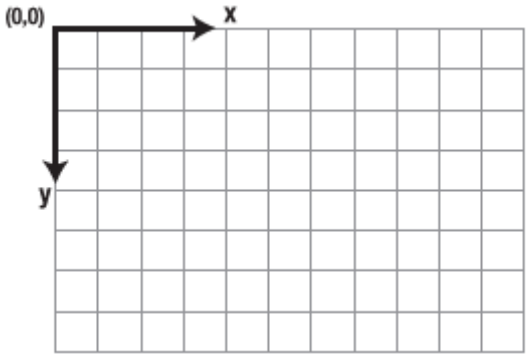


Fig. 24: Sistema de coordenadas utilizado en canvas.

Estos valores tienen, además, un significado especial, porque si reasignamos cualquiera de ellos después de haber realizado cualquier operación, borrará todo su contenido y reiniciará las propiedades del contexto de dibujo (incluso si se reasigna al mismo valor que tienen en ese momento).

Las instrucciones de dibujo, se programarán, por tanto, mediante código JavaScript en la manera habitual. Si queremos tener una primera idea de cómo quedaría un dibujo, podemos complementarlo con un estilo que defina un borde, y comenzar por la primera instrucción básica, la de la construcción de un trazo mediante el movimiento desde un punto inicial a un punto final:

```
    <h4>Demo de Canvas - Pulse sobre la superficie para
dibujar</h4>
   <canvas id="Canvas1" width="300" height="200"
       style="border:3px solid;"
onclick="dibujoBasico();" />
   <script>
       function dibujoBasico() {
           // Obtenemos la referencia al elemento y
        // su contexto.
        var canvas = document.getElementById('Canvas1');
        var context = canvas.getContext('2d');
```

```
    // Para dibujar una línea, nos movemos a punto
    //inicial y utilizamos el método lineTo()
         context.beginPath();
         context.moveTo(20, 20);
         context.lineTo(280, 150);
         // Dibujar la línea
         context.stroke();
         context.closePath();
        }
    </script>
```

Lo que genera una salida (idéntica en todos los navegadores analizados, como la de la figura 25:

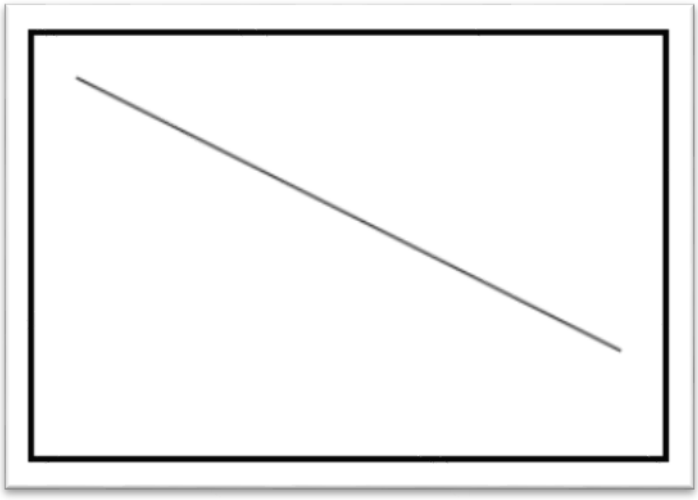


Fig. 25: Salida en IE9/10/11 del código anterior, mostrando el elemento <canvas>, tras hacer clic sobre la página.

Como vemos, no solo es necesario definir los puntos de origen y destino de la línea, sino que tenemos que indicarle explícitamente que lo dibuje (método *Stroke*). El proceso es similar a cuando hacemos un esquema a lápiz sobre una superficie, y finalmente, decidimos validar el esquema dibujando en tinta encima de los trazos. Así es como funciona el objeto **Path**, que solo se muestra después de una operación ***stroke*** (dibujar) o ***fill*** (rellenar).

Respecto a la instrucción ***getContext('2D')***, hay que mencionar que todo lo que hagamos será en un entorno bidimensional, pero existe un proyecto para poder extender las capacidades de **canvas** a 3 dimensiones, así como simulaciones en ese sentido. (En la web pueden encontrarse varios ejemplos disponibles).

Rellenos y otras composiciones

Continuando con nuestro ejemplo anterior, podríamos dibujar un triángulo mediante la adición de este par de líneas al código anterior:

```
// Añadido para dibujar un rectángulo
context.lineTo(20, 150);
context.lineTo(20, 20);
```

Y si queremos rellenar la superficie definida, solo tendremos que llamar al método *fill*, como hemos indicado, y si no queremos que los colores sean los predeterminados (relleno en negro), podemos completar el código con la definición de esas propiedades mediante los métodos *fillStyle* y *strokeStyle* que establecen los colores de relleno y borde respectivamente. Admiten como propiedades cadenas que definen los colores correspondientes, como "#ccc" (tipo de gris para el interior), o "*Navy*" (azul oscuro) para el borde del rectángulo. Si queremos resaltar más el borde podemos utilizar también *liidth* para cambiar el grosor. Con estas 3 líneas de código nuevas para nuestro rectángulo, obtendríamos una salida como la de la figura 26.

```
// Para dar formato al rectángulo
context.fillStyle = "#ccc";
context.strokeStyle = "Navy";
context.lineWidth= 9;
```

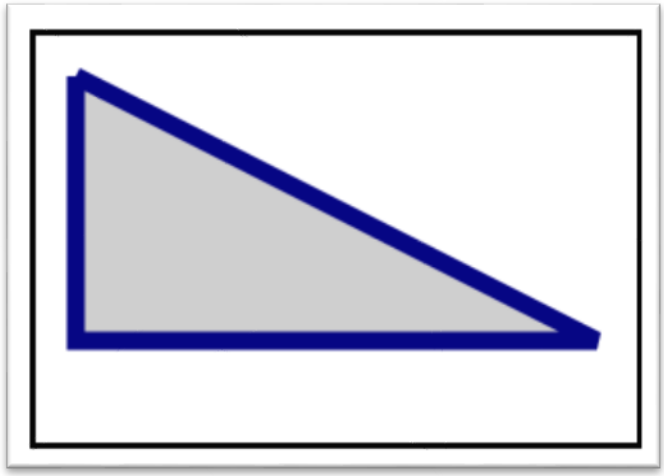


Fig. 26: el rectángulo mostrando sus bordes, y relleno interior en los colores definidos.

> **Nota**: Cuando un objeto Path se cierra mediante una instrucción closePath(), es posible rellenar su contenido, incluso si no se ha trazado la última línea que cierre la figura. closePath es también un indicador de terminación de la sesión de dibujo.

La propiedad ***lineJoin*** puede valer: ***miter***, ***round*** y ***bevel***, mientras que ***lineCap*** puede adoptar los valores ***butt, square*** y ***round***. Su significado, queda mucho más claro en su gráfico como el de la figura siguiente.

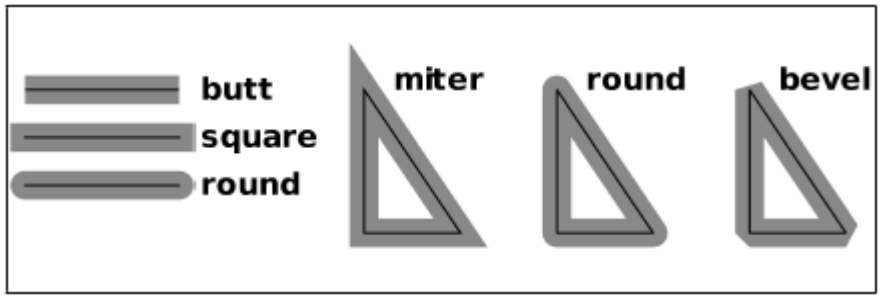


Fig. 27: Valores y significados de las propiedades lineCap y lineJoin.

Podríamos añadir un par de líneas a nuestra definición anterior configurando esos valores:

```
context.lineJoin='bevel';
context.lineCap='round';
```

Y los resultados son los que aparecen en la figura 28:

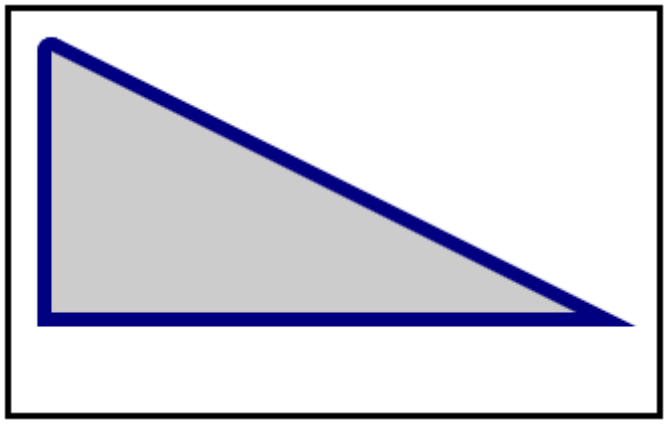


Fig. 28: La misma figura anterior, con un factor de corrección lineJoin

Dibujando otras figuras básicas

Naturalmente el API de **canvas** ofrece otras posibilidades relacionadas con el dibujo de las figuras más habituales, como las figuras geométricas más conocidas (rectángulos, círculos, elipses, etc.). Por ejemplo, si queremos crear un rectángulo relleno en rojo, el código siguiente bastará para la tarea:

```
        // Definir un rectángulo azul
        context.fillStyle = "Navy";
        context.fillRect(20, 40, 250, 150);
```

Donde los puntos (20,40) Y (250,150) definen, respectivamente los dos extremos de la diagonal del rectángulo a trazar.

Sin embargo, no existe una instrucción *fillEllipse*, como cabría esperar, sino que esa operación se realiza mediante la sentencia dedicada al

trazado de arcos de circunferencia, que tiene la siguiente declaración formal:

```
context.arc(x, y, radius, startAngle, endAngle,
counterClockwise)
```

Aquí *x* e *y* son las coordenadas del centro, *radius* el radio, y *startAngle* y *endAngle* los ángulos inicial y final, expresados en radianes. Finalmente, *counterClockwise* es un valor *booleano* que indica si el relleno del círculo (del arco en realidad), debe realizarse en el sentido de las agujas de reloj o en el contrario. Por tanto, si queremos trazar un círculo relleno con un borde dentro de nuestro rectángulo, deberemos añadir las siguientes líneas al código anterior:

```
    // Definir un circulo, rellenarlo
    // y dibujarlo
    context.fillStyle = "Yellow";
    context.arc(150, 100, 50, 0, Math.PI * 2, false);
    context.fill();

    // Definir el borde del círculo con un grosor,
    // rellenarlo y dibujarlo
    context.arc(150, 100, 50, 0, Math.PI * 2, false);
    context.strokeStyle = "Orange";
    context.liidth = 9;
    context.stroke();
```

Y, como es de esperar, se genera una salida será como la de la figura 29:

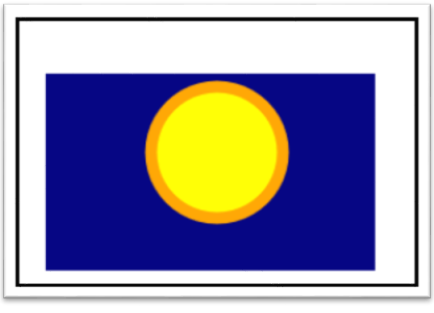


Fig. 29: trazado de figuras geométricas con el API de canvas.

Hay que notar que cada "sesión de trazado", o sea, cada dibujo individual, conlleva una llamada a la instrucción encargada de ello: dos llamadas a *fill*, para las rellenar el rectángulo y el círculo, y una llamada a *stroke* para el borde, que es equivalente a trazar la circunferencia correspondiente a ese círculo, con un grosor de trazo de 9 píxeles.

Con el trazado de arcos pueden conseguirse figuras más complejas, pero el API también suministra un mecanismo de trazado de curvas suaves (Bézier), que está disponible en sus variedades cuadrática y curva, mediante las funciones **`quadraticCurveTo(`**`cp1x, cp1y, x, y)`**`, y bezierCurveTo`** `(cp1x, cp1y, cp2x, cp2y, x, y)`, respectivamente.

> **Nota**: Recordemos que una curva cuadrática se define mediante dos puntos de anclaje (anchor points), y un punto de control (control point), que define al curvatura entre los dos anteriores (como los paraboloides). Por su parte, una curva cúbica tiene dos puntos de control, lo que nos permite dibujar p.ej. curvas con un punto de inflexión.

Vamos a crear un nuevo dibujo que implemente estos dos tipos de curvas en el mismo **canvas**:

```
// Empezamos por situarnos en un punto inicial
context.beginPath();
// Circunferencia amarilla
context.fillStyle = "Yellow";
context.arc(150, 100, 50, 0, Math.PI * 2, false);
context.fill();
// Circunferencia naranja
context.arc(150, 100, 50, 0, Math.PI * 2, false);
context.strokeStyle = "Orange";
context.lineWidth = 25;
context.stroke();
context.closePath();
```

```
// Curvas Bezier y Cuadrática
context.beginPath();
context.strokeStyle = "Blue";
context.lineWidth = 19;
context.moveTo(0, 0);
context.quadraticCurveTo(100, 25, 50, 50);
context.stroke();
context.closePath();

context.beginPath();
context.strokeStyle = "Green";
context.moveTo(150, 0);
context.bezierCurveTo(0, 125, 200, 75, 150, 200);
context.stroke();
context.closePath();
```

Al final, obtendremos una salida como la de la figura 30:

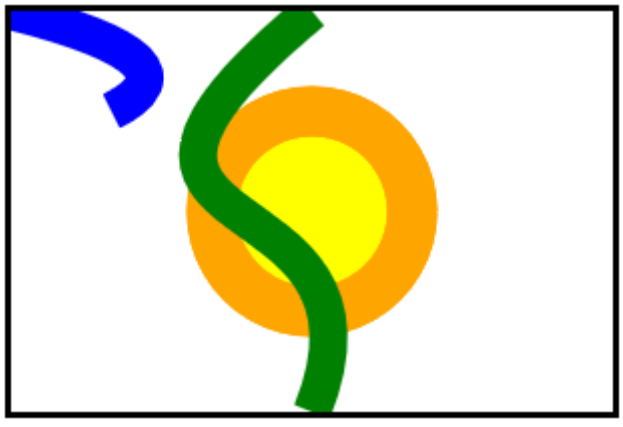


Fig. 30: Trazado de dos curvas Bézier de tipo cuadrático y cúbico.

Las posibilidades con estas opciones son enormes, y similares a las que podemos encontrar en sus contrapartidas propietarias Flash y Silverlight.

Gradientes de color

El uso de gradientes de relleno (gamas de color que van progresando desde un valor inicial a otro final y pueden tener una serie de puntos

intermedios de cambio) es habitual en todas las herramientas de dibujo actuales.

En general, podemos distinguir dos tipos: gradientes radiales y lineales, dependiente del modo de dibujo utilizado. El API de **canvas** lo implementa mediante las funciones *createLinearGradient* y *createRadialGradient*. Podemos crearlos directamente, de manera uniforme, indicando un punto de origen y un punto final, o podemos intercalar entre el origen y el final un número indeterminado de puntos de parada (***colorStop***), siendo el mínimo de dos, indicando únicamente los puntos inicial y final.

Por ejemplo, si queremos definir un gradiente lineal, para rellenar un rectángulo que vaya desde el color amarillo hasta el azul, primero declararemos el gradiente a utilizar y lo recogeremos en una variable para poder manipularlo posteriormente:

```
/* Definición del gradiente Lineal */
var gradient = context.createLinearGradient(0, 0, 300, 50);
/* Puntos de inicio y fin */
gradient.addColorStop(0, "Yellow");
gradient.addColorStop(0.25, "Green");
gradient.addColorStop(0.6, "Lime");
gradient.addColorStop(1, "Blue");
/* Asignación y proceso de relleno */
context.fillStyle = gradient;
context.fillRect(0, 0, 300, 200);
```

Donde las coordenadas (0,0) y (300,200) se corresponden con las de la superficie de nuestro elemento **canvas.** Esto nos crea un mecanismo de relleno, pero tendremos que indicar de qué color a qué color varía y cuáles son los puntos de inicio y fin.

Lo que nos generará una salida como la de la figura 31:

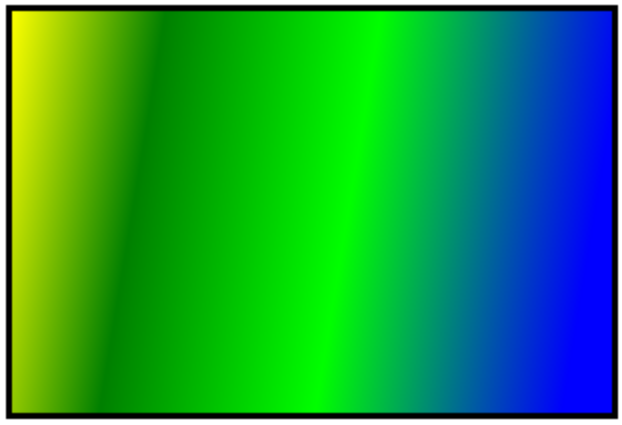


Fig. 31: gradiente lineal sobre toda la superficie del canvas

De la misma forma, podemos crear un gradiente radial a partir de un punto y definir un círculo con varios "anillos" de color, podríamos utilizar el código siguiente:

```
var gradient = context.createRadialGradient(100, 50, 25,
100, 100, 100);
// Define los colorStop
gradient.addColorStop(0.2, 'Red');
gradient.addColorStop(.66, 'Blue');
gradient.addColorStop(1, 'Yellow');
// Define el estilo de relleno usando el gradiente
anterior
context.fillStyle = gradient;
context.arc(100, 100, 100, (Math.PI / 180) * 0, (Math.PI
/ 180) * 360, false);
context.fill();
```

Y obtendríamos una salida como la de la figura 32:

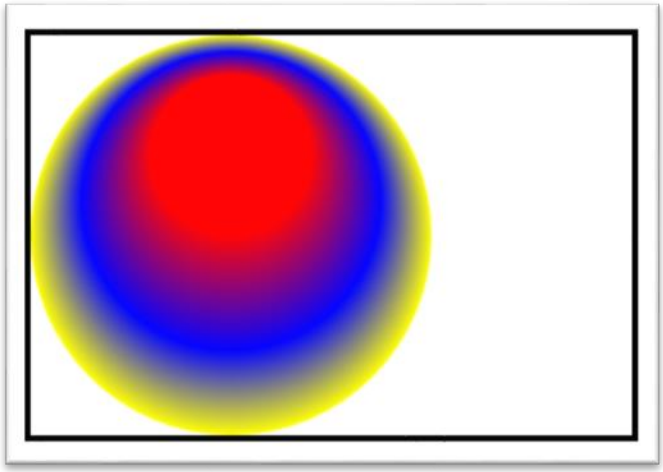


Fig. 32: Gradiente radial sobre un círculo.

Trabajando con texto

Lo que hemos hecho hasta ahora es dibujar, o sea utilizar las primitivas del API de **canvas** para generar diversas figuras geométricas. Pero en el uso real, lo normal es que combinemos estas características con texto e imágenes, esto es, incorporando elementos ya construidos (letras o contenido en formato .jpg o .png) dentro de nuestro **canvas**.

Para dibujar texto, lo primero que tenemos que tener en cuenta es que no se trabaja como con los elementos HTML habituales, sino más bien, como se usa el texto en las herramientas de diseño. Esto es, definimos un tipo de letra, un texto a dibujar, y una posición en el **canvas**, y ahí se comienza la escritura.

Una vez más, el objeto *context* nos suministra los elementos necesarios para ello: *font,* para indicar el tipo de letra (con todas sus variantes de estilo, resaltado, etc.), *textAlign*, para la alineación, *textBaseline*, para establecer en qué punto empezamos a dibujar respecto al punto de origen (recomendamos echar un vistazo a la especificación para más detalles sobre los distintos valores posibles para este atributo), y, finalmente, *fillText*, para generar la salida del texto en pantalla. Esta función tiene la siguiente declaración formal:

```
fillText([text],[x],[y],[maxWidth]);
```

Si queremos dibujar el normativo "*Hello World*" en nuestro **canvas**, con un tipo de letra Segoe UI de 18 puntos, en la parte superior izquierda, necesitaríamos el siguiente código:

```
context.font = 'bold 28px Segoe UI';
context.fillText('Hello World', 50, 50);
```

También podemos conseguir efectos curiosos mediante la función *strokeText*, que solamente dibuja el borde de la letra (especialmente interesante cuando trabajamos con tamaños grandes), pudiendo utilizar para el relleno la función *fillText*. Por ejemplo, el código siguiente, dibuja la palabra "Texto <canvas>", con un borde azul y un relleno en rojo, siendo el tamaño de letra de 52 puntos:

```
context.font = 'bold 168px Segoe UI';
context.lineWidth = 8;
context.strokeStyle = "Navy";
context.fillStyle = 'Red';
context.shadowColor = "rgba(100,100,100,.4)";
context.shadowOffsetX = context.shadowOffsetY = -10;
context.shadowBlur = 5;
context.strokeText('Canvas', 50, 140);
context.fillText('Canvas', 50, 140);
```

Por tanto dibujamos el texto dos veces. Una para el borde y otra para la parte interior delimitada por éste. La salida es como la de la figura 33:

Fig. 33: dibujo de un texto con efectos de relleno.

Donde la salida generada es prácticamente idéntica en todos los navegadores analizados.

También hemos aplicado un efecto especial, como es el sombreado. Para crear un sombreado, operamos como siempre. Primero, definimos un color para la sombra (*shadowColor*), un desplazamiento u *offset* (*shadowOffset,* la cantidad de sombra), y un nivel de nitidez (*shadowBlur*), estos dos últimos medidos en píxeles.

Podemos jugar con esos parámetros para cambiar la orientación de la sombra (el origen de la luz). Así pues, nos basta con cambiar el siguiente código de definición de sombra *antes* de las líneas anteriores:

```
// Definición de la sombra
context.shadowColor = "rgba(100,100,100,.4)";
context.shadowOffsetX = context.shadowOffsetY = 12;
context.shadowBlur = 5;
context.strokeText('Texto <canvas>', 50, 140);
```

Y tendremos una salida como la de la figura 34:

Fig. 34: El mismo texto anterior, pero con un efecto de sombreado

Utilizando imágenes en un <canvas>

Respecto a la carga y manipulación de imágenes, el procedimiento es, hasta cierto punto, similar al del trabajo con texto. El primer paso es

obtener una referencia al gráfico a presentar. Este puede ser un elemento **<img>** del código de marcado, o puede crearse dinámicamente mediante una llamada al método "constructor" del objeto *Image* de este API. Una vez hecho esto, tendremos que asignarle una imagen real, en su propiedad *src*, que tendrá que apuntar a una URL válida (local o remota).

El último paso es llamar al método *DrawImage* del objeto *context*, indicándole cuál es la imagen, y el resto de parámetros a utilizar, que en este caso, son la posición donde se comienza a dibujar la imagen, y el tamaño deseado (anchura y altura).

Una nota sobre el soporte de V. Studio 2012

Además, como podemos observar en la figura siguiente, según trabajamos con las instrucciones de dibujo, Visual Studio 2012 nos va asistiendo en la tarea recordándonos, por ejemplo, el número de sobrecargas de que dispone cada uno de los métodos, tal y como sucede en este caso.

El método ***drawImage***, no solo puede usarse para gráficos creados dinámicamente, sino con etiquetas ***<img>*** de HTML 5 declaradas en otra parte de la página, o incluso con vídeos, definidos mediante etiqueta HTML, de la misma forma. Podemos ver este soporte en la figura siguiente:

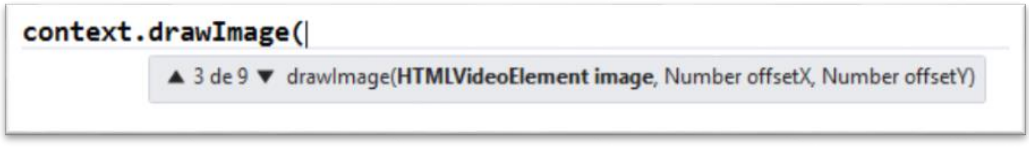


Fig. 35: Soporte del Editor de código de Visual Studio 2012, para el elemento <canvas>

Lo mismo podemos hacerlo extensible a todas las otras API del estándar, como iremos viendo a lo largo del libro al abordar cada una de ellas.

El siguiente código consigue el objetivo básico de dibujar una imagen local, generando una salida como la de la figura 36:

```
// Creación del objeto para dibujar
var imagen = new Image();
// Asignación del dibujo a utilizar
imagen.src = 'Graficos/Tintin.png';
// Dibujo la posición 10,15
context.drawImage(imagen, 10, 15, 200, 200);
```

Fig. 36: Dibujando una imagen PNG en un canvas.

Donde los parámetros segundo y tercero, establecen las coordenadas a partir de las cuales se comienza a dibujar la imagen. Opcionalmente, un cuarto y un quinto parámetros, corresponden al tamaño deseado (si no se indica, como en este caso, se usa el predeterminado).

También podemos utilizar una imagen existente para realizar copias o variantes de la misma. Para ello le indicamos una zona rectangular que nos servirá de punto de partida, y otra que será la de destino. En ambos casos, debemos indicar punto de origen, anchura y altura, por lo que podremos tener copias a diferentes resoluciones.

Si añadimos a continuación de la sentencia anterior de dibujo esta otra, obtendremos dos imágenes, situadas en distintas partes del **canvas**, y con distintas resoluciones, como aparece en la Fig. 37:

```
// Dibujo con escala al lado del original
// y cambiamos la imagen por otra
context.drawImage(imagen, 0, 0, 255, 197, 300, 50, 125, 100);
```

Fig. 37: Salida del código anterior en IE11

Aquí la diferencia es que la segunda sentencia establece que va a dibujar la imagen inicial completa (podría leer solamente una zona indicada por los 4 parámetros de coordenadas iniciales), y se va a replicar en otra zona, pero, como establecemos unos parámetros de superficie, se produce un efecto de "*clipping*" (recorte) de la imagen.

Transformaciones

Al igual que sucede con las API de Flash y Silverlight, el API de **canvas** ofrece la posibilidad de realizar transformaciones sobre las imágenes o sus definiciones. Esas transformaciones son fundamentalmente de 3 clases (al no ser de tipo vectorial, el usuario tendrá que valorar el factor de pérdida de calidad correspondiente):

- **Escalado**: (*scale(x, y)* similar al anterior, solo que los valores son relativos a 1 que es el tamaño original: Ej.: 1.20 es un incremento del 20% sobre el total)
- **Rotaciones**: (*rotate(angle),* siendo *angle el* ángulo expresado en radianes, mediante la fórmula: *radianes = grados sexagesimales * Pi / 180*)
- **Traslaciones**: (*translate(x,y)*, que indican desplazamiento, indicando las dos coordenadas de la ubicación de destino)

Podemos utilizar cualquiera o todas ellas aplicándolas una a continuación de otra, o podemos utilizar el método *transform*, que permite indicar conjuntos de transformaciones que se aplican de forma simultánea a un objeto, indicando los valores a transformar en el mismo orden que el indicado en la lista (escalado, rotaciones, traslación). El método trabaja sobre una matriz de transformación muy similar a las que usan otros entornos de dibujo como Silverlight. El código sería algo como lo siguiente:

```
context.transform(escalaX, escalaY, rotX, rotY, dx, dy);
```

También tenemos que tener en cuenta que cualquier transformación definida no se aplica realmente hasta que una función de transformación no es llamada realmente, y que las transformaciones se aplican incrementalmente. Esto quiere decir que los nuevos cálculos siempre se añaden a los anteriores, a no ser que se reinicie la matriz de transformación, con una instrucción como:

```
context.setTransform(1, 0, 0, 1, 0, 0);
```

Veremos algo más de la puesta en práctica de estas técnicas a continuación, en el apartado de animaciones y contenido dinámico.

Contenido dinámico: Animaciones en *canvas*

Comencemos por la definición. Entendemos por animación una ilusión

visual, creada mediante la proyección rápida y continua de varias imágenes o gráficos, cada uno de ellos ligeramente distinto del anterior. Las técnicas para la creación de vídeos no son más que un sistema de animaciones de fotogramas.

En HTML 5 no existe el concepto de *StoryBoard,* como sucede en Silverlight o Flash. Eso quiere decir que las animaciones son simplemente secuencias de movimiento generadas por nosotros mediante un temporizador que realiza una llamada a una función que tendrá como cometido realizar esa presentación de imágenes, cada una distinta de la anterior, como indicábamos en la definición.

Por tanto, la técnica para crear animaciones puede consistir en crear una imagen, e indicarle a un temporizador que llame a una función que vuelva a pintar dicha imagen en una posición desplazada de la primera. En nuestro ejemplo, comenzamos por pintar nuevamente nuestra imagen anterior, pero utilizamos una función para calcular una rotación, de forma que –cada vez que se llama- el dibujo vuelve a repintarse después de haber girado una cantidad concreta.

El código fuente siguiente, consigue ese efecto, si bien como vuelve a pintar desde la misma posición inicial, obtenemos un efecto circular de superposición (Ver fig. 38):

```
<!DOCTYPE HTML>
<html>
<head>
<title>Animaciones</title>
<script type="text/javascript">
    var drawing =  Image();
    drawing.src = 'Graficos/ie.png';

    function establecerAnimacion() {
        setInterval(animarDibujo, 100);
    }

    function animarDibujo() {
        // Cada décima de segundo se recupera
        // el canvas y el contexto de dibujo
```

```
        var canvas = document.getElementById('Canvas1');
        var ctx = canvas.getContext('2d');
        // Se guarda el estado inicial del contexto
        // para restaurarlo después de dibujar
        ctx.save();
        // Desplazamiento al punto inicial
ctx.translate(200, 200);
// La rotación depende del tiempo transcurrido
var time = new Date();
ctx.rotate(((20 * Math.PI) / 180) * time.getSeconds());
ctx.translate(0, 28.5);
ctx.drawImage(drawing, -1, -1, 100, 100);
// Recuperamos el contexto (implica borrado)
ctx.restore();
    }
</script></head>
<body onload="establecerAnimacion();">
    <canvas id="Canvas1" width="600" height="480">
    </canvas>
</body>
</html>
```

Lo que produce una salida como la de la figura siguiente (la captura de pantalla podría no ser exactamente igual en su caso).

Fig. 38: Salida del código anterior, resultado de una animación de rotación.

Si queremos que la acción represente una auténtica animación (sensación de movimiento), solo tenemos que añadir una instrucción de borrado de la superficie que usamos. Por ejemplo de toda la zona del *canvas*, añadiendo antes del método *save()*:

```
ctx.clearRect(0, 0, 600, 480);
```

Y obtendríamos una animación que se desplaza cada fracción de segundo.

Naturalmente, esta forma de hacer animaciones requiere bastante trabajo cuando las animaciones son más complejas. Debido a eso están apareciendo varias utilidades en la red que tiene ese propósito: facilitar la construcción de animaciones de forma visual, al estilo de lo que podemos hacer hoy con las herramientas de Silverlight (*Expression Blend*) o *Adobe*.

Otro de los efectos más habituales es el de mezclar las animaciones y los temporizadores con la manipulación de imágenes para crear efectos visuales espectaculares y/o sensación de "recorrido" de un contexto.

Efectos de animación sobre imágenes

Muchos de los efectos habituales que vemos en juegos, en la Web y otros contextos se basan en los principios de gestión de imágenes en movimiento. Existen librerías especializadas en todos estos temas, y no vamos a profundizar más en el tema, pero puede ser interesante ver cómo podemos utilizar ***canvas*** para programar un desplazamiento con un control básico de colisiones.

Para ellos, planteamos aquí el típico ejemplo de un círculo encerrado en una superficie rectangular, que va rebotando al llegar a sus bordes.

La parte HTML es trivial:

```
<canvas id="canvasCirculo" width="300" height="300">
    Su navegador no soporta HTML 5 Canvas.
</canvas>
```

En la parte JavaScript, el siguiente código se encarga de las definiciones del contexto y el círculo, de ponerlo en marcha una vez cargado, de controlar el dibujo del círculo cada 25 ms., y de comprobar la posición para detectar la colisión con los bordes del contenedor.

```
<script>
    window.addEventListener('load', iniciar, false);
    function iniciar() {

        var velocidad = 5;
        var p1 = { x: 20, y: 20 };
        var angulo = 35;
        var radianes = 0;
        var unidadesX = 0;
        var unidadesY = 0;
        var radio = 20;
        var circulo = { x: p1.x, y: p1.y };

        function dibujar() {
            context.fillStyle = '#EEEEEE';
            context.fillRect(0, 0, lienzo.width,
lienzo.height);
            //Contenedor
            context.strokeStyle = '#000000';
            context.strokeRect(1, 1, lienzo.width - 2,
lienzo.height - 2);
            circulo.x += unidadesX;
            circulo.y += unidadesY;
            context.fillStyle = "navy";
            context.beginPath();
            context.arc(circulo.x, circulo.y, radio, 0,
Math.PI * 2, true);
            context.closePath();
            context.fill();
            if (circulo.x + radio > lienzo.width ||
circulo.x < radio) {
                angulo = 180 - angulo;
                dibujarCirculo();
            } else if (circulo.y + radio > lienzo.height
|| circulo.y < radio) {
```

```
                    angulo = 360 - angulo;
                    dibujarCirculo();
                }
            }

            function dibujarCirculo() {
              radianes = angulo * Math.PI / 180;
              unidadesX = Math.cos(radianes) * velocidad;
              unidadesY = Math.sin(radianes) * velocidad;
            }

            dibujarCirculo();
            lienzo =
document.getElementById('canvasCirculo');
            context = lienzo.getContext('2d');
            setInterval(dibujar, 25);
        }
    </script>
```

Al ponerlo en ejecución, tendremos el clásico escenario que hemos visto tantas veces de una bola rebotando en los límites de un cuadrado, como se muestra en la figura 39:

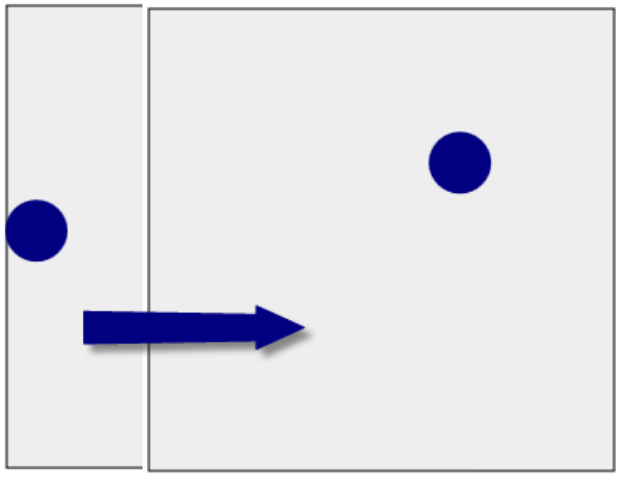


Fig. 39: El código fuente anterior, en funcionamiento.

El elemento <*svg*>

Además, la sintaxis HTML de HTML5 permite la inclusión de elementos SVG[31] (*Scalable Vector Graphics*), una especificación para describir gráficos vectoriales bidimensionales, tanto estáticos como animados (estos últimos con ayuda de SMIL), en formato XML.

Fig.40: Logo de SVG que se muestra en la página oficial de la W3C

SVG no es un estándar de HTML5. Su mantenimiento es llevado a cabo por la W3C, y el estándar soporta una etiqueta configurable (**<*svg*>**), que permite utilizarlo como mecanismo de creación de gráficos vectoriales.

Prácticamente, todos los navegadores ofrecen soporte de éste lenguaje (puede verse mucha más información relacionada en el enlace a pie de página). También recomendamos un vistazo a la versión actualizada del borrador, (*Editor's Draft*)[32].

SVG permite tres tipos de objetos gráficos[33]:

- Formas gráficas de vectores (p.ej., caminos consistentes en rectas y curvas, y áreas limitadas por ellos)

[31] La página oficial del estándar SVG publicada en la W3C es http://www.w3.org/Graphics/SVG/

[32] https://svgwg.org/svg2-draft/Overview.html

[33] Existe una página explicativa de los orígenes de este lenguaje titulada "*The Secret Origin of SVG*" en http://www.w3.org/Graphics/SVG/WG/wiki/Secret_Origin_of_SVG

- Imágenes de mapa de bits /digitales
- Texto

SVG es un lenguaje de marcado para la descripción de aplicaciones con gráficos bidimensionales, imágenes y un conjunto de interfaces de secuencia de comandos para gráficos relacionados. SVG 1.1 es una recomendación del W3C y es la versión más reciente de la especificación completa; una segunda edición de SVG 1.1 que incluye aclaraciones y mejoras menores, según los comentarios de desarrollador se encuentra actualmente en estado "*Last Call*". Además, para dispositivos móviles, existe SVG Tiny 1.2, que es igualmente, una recomendación del W3C.

SVG 2 está actualmente en desarrollo y agrega nuevas características de facilidad de uso para SVG, así como una más estrecha integración con HTML, CSS y el DOM.

El grupo de trabajo de SVG está trabajando actualmente en paralelo en un conjunto de módulos para extender las especificaciones anteriores y agregar funcionalidad a CSS, y la nueva especificación SVG 2 combinará esos módulos con el resto del marco SVG para poder trabajar a través de toda la gama de dispositivos y plataformas.

No hace mucho, la W3C publicaba un documento de convergencia de estándares que recoge las propuestas para una nueva especificación que aúna las definiciones *CSS Transforms* (2D y 3D) junto a las transformaciones para SVG, en un solo bloque[34].

De cara a la implementación en navegadores, la W3C publicó un documento en Septiembre 2010 ("*An SVG Primer for Today's Browsers*")[35], en calidad de *W3C Working Draft*, que está sirviendo de "documento de análisis" a los fabricantes de navegadores.

A modo de ejemplo simple (volveremos sobre el tema más adelante), digamos que el código siguiente basta para dibujar un círculo azul de 60

[34] http://dev.w3.org/csswg/css3-transforms/ (de fecha 1-junio-2012).

[35] http://www.w3.org/Graphics/SVG/IG/resources/svgprimer.html

píxeles de radio:

```
<!doctype html>
  <title>SVG in text/html</title>
  <p> A blue circle:
    <svg>
    <circle r="60" cx="60" cy="60" fill="blue"/>
    </svg>
  </p>
```

Lo que produce la siguiente salida en Internet Explorer 9/10//11:

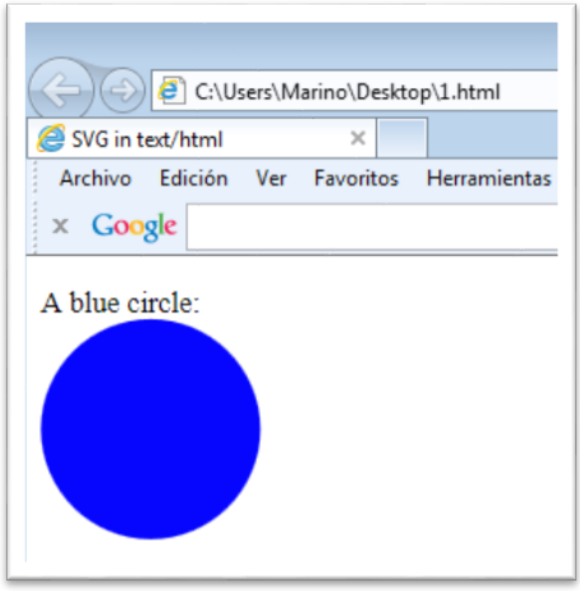


Fig.41: Salida del código anterior en IE9/10/11

La etiqueta ***<svg>*** permite definir, además el tamaño exacto que queremos utilizar para la superficie de dibujo, y qué zona de ese tamaño definido se utilizará para la presentación gráfica:

```
<svg width="11.3cm" height="2.2cm" viewbox="0 0 1130 220"
version="1.1" xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink">
```

Dentro de una etiqueta ***<svg>*** existe un conjunto de etiquetas de tipo estructural que ayudan en la construcción del ***layout*** o disposición visual de los elementos. Uno de los más importantes es ***<g>***, que permite agrupar diversos elementos en un bloque, de forma que se les pueda aplicar posteriormente efectos u otras instrucciones de forma colectiva. De hecho, estas agrupaciones se entienden como definiciones previas, que son "dibujadas" a posteriori mediante otro elemento <g>, por ejemplo, que establezca una ubicación inicial y un vínculo para referenciar su nombre.

Por ejemplo, el código siguiente agrupa 3 figuras geométricas en un bloque, para dibujarlas o para aplicar transformaciones más adelante.

```
<g id="CirculoRectanguloTriangulo">
   <g>
    <circle cx="85" cy="80" r="40" fill="yellow" />
    <g>
    <rect x="145" y="45" width="90" height="70" fill="red" />
    </g>
    <path d="M260,40 L335,80 L260,120 z" fill="#4444ff" />
   </g>
</g>
```

Observe que, al igual que en otros lenguajes de definición de interfaces de usuario y gráficos, como XAML, se dispone de un elemento **Path**, que utiliza el lenguaje M definido para ellos, para poder dibujar trazos de cualquier clase de manera vectorial. En este caso, tras el círculo y el rectángulo, se dibuja un triángulo, estableciendo un punto inicial, y 3 trazos dentro de una línea cerrada, rellenos con un color azul (atributo ***fill***).

Más adelante, podemos añadir otro elemento ***<g>***, que sitúe esa definición gráfica en cualquier parte de la superficie de dibujo:

```
<g>
<use x="150" y="50"
xlink:href="#CirculoRectanguloTriangulo" />
</g>
```

Ese código que genera una salida como la siguiente:

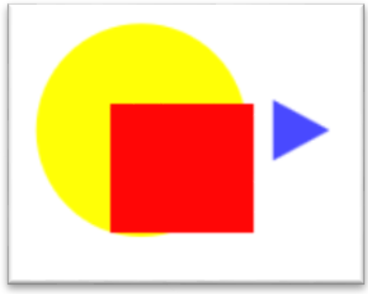


Fig. 42: Salida del código anterior en IE10/11.

El usuario que haga pruebas de SVG con las versiones de Visual Studio 2012, (o 2013 *Preview*) observará que disponemos de un amplísimo soporte de este estándar (que, sin serlo directamente, se relaciona con HTML5).

Así, el editor nos ofrece soporte completo de los elementos que tiene sentido empotrar dentro de un par de elementos ***<svg></svg>***, así como de los atributos que cada uno de los elementos de SVG pueden adoptar en un momento dato, ofreciendo una experiencia de desarrollo habitual en Visual Studio como podemos ver en las siguientes figuras, que muestran el soporte de distintos niveles de anidación de elementos (el principal y dependiente de directos de ***<svg>***, y los posibles atributos de cada uno de los elementos SVG empotrados, como es el caso de ***<filter>***.

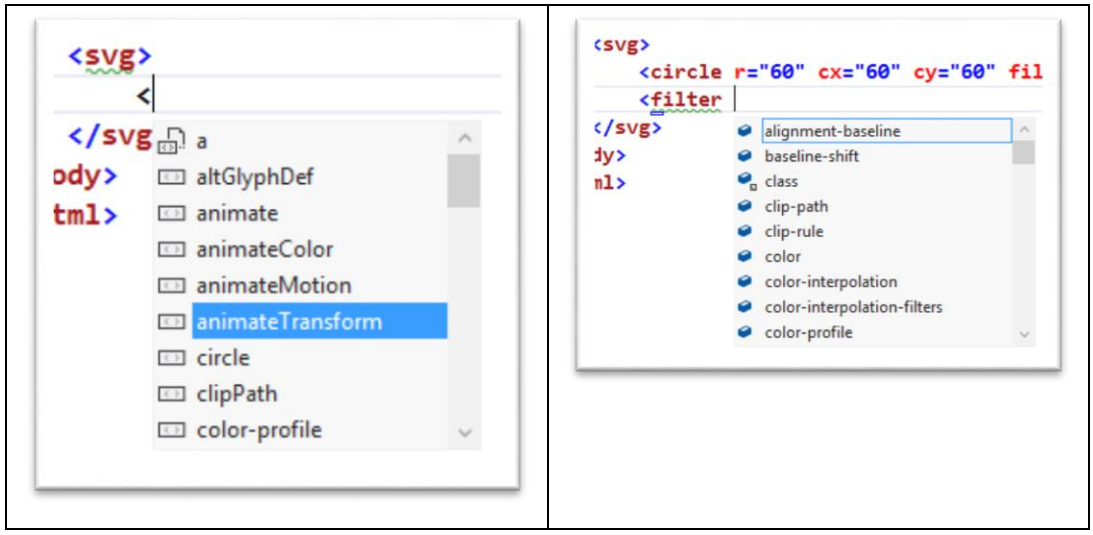


Fig. 43: Soporte del estándar SVG en el editor de código Visual Studio 2012/2013

SVG y los filtros

Una de las características más interesante de este tipo de gráficos es la denominada SVG *Filters*, que permite realizar determinadas operaciones sobre gráficos. Son de naturaleza declarativa son etiquetas, tienen soporte del DOM (para su manipulación mediante JavaScript), y pueden aplicarse de dos formas: mediante el atributo ***filter*** del elemento, o mediante estilos CSS, en el atributo de igual nombre, haciendo referencia a éste por su identificador: **`filter:url(#filterId)`**. Por ahora, nos limitaremos a aquellas que siguen la sintaxis nativa HTML. Para más datos sobre los tipos de filtros disponibles ver la página oficial del estándar SVG[36]. Todos ellos están siendo soportados (en edición, mediante *Intellisense*), y en tiempo de ejecución, por IE10.

Siquiera por encima, ya que no es ese el objetivo de esta obra, comentaremos que cada filtro consta de una serie de "primitivas de filtrado", que son declaraciones que permiten aplicar diferentes efectos a los gráficos, y se aplican posteriormente. Además, para formar filtros

[36] http://dev.w3.org/SVG/profiles/1.1F2/publish/filters.html

complejos, esas primitivas pueden encadenarse.

Por ejemplo, el siguiente código, añadido al anterior, produce un efecto de rotación de color (***hueRotate***), en el elemento anterior, convirtiéndolo en un círculo de color rojo, a partir de una primitiva ***feColorMatrix***:

```
<circle filter="url(#rotacionColorHue)" r="60" cx="60"
cy="60" fill="blue" />
<filter id="rotacionColorHue">
    <feColorMatrix type="hueRotate" values="90" />
</filter>
```

Lo que, lógicamente produce la misma salida (fig. 4), pero modificada (nótese que podríamos hacer dicho filtro se produjese a lo largo de un intervalo de tiempo, con lo que conseguiríamos un efecto de animación de color:

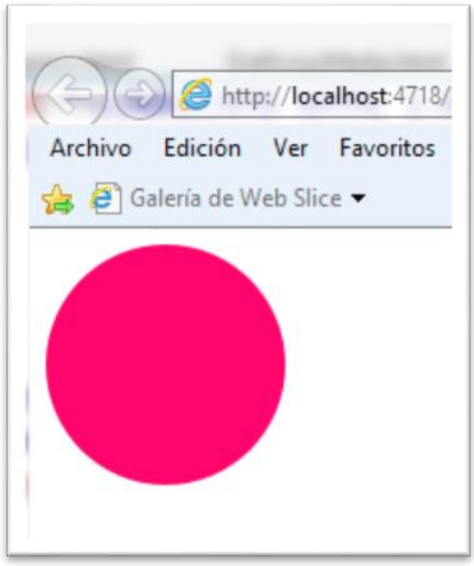


Fig. 44: salida del código anterior después de la transformación de color por filtro

Por supuesto, el estándar también permite cargar imágenes de recursos externos o locales, habiendo añadido previamente un espacio de nombres, que define los atributos necesarios. De acuerdo con eso, podríamos hacer la misma transformación, aplicándola esta vez a un

gráfico local, gracias a las siguientes modificaciones al código fuente anterior:

```
<svg xmlns="http://www.w3.org/2000/svg"
         xmlns:xlink="http://www.w3.org/1999/xlink">
   <filter id="rotacionColorHue">
      <feColorMatrix type="hueRotate" values="90" />
   </filter>
   <image xlink:href="Graficos/HTML5_Logo.png" x="0"
y="0" height="250px" width="250px"
filter="url(#rotacionColorHue)">
   </image>
</svg>
```

Es interesante notar que la etiqueta ***image*** de SVG requiere obligatoriamente la indicación de los parámetros de anchura y altura.

Mediante esa definición y la aplicación de un filtro idéntico al anterior, la salida en el navegador IE10, tendría el siguiente aspecto (Ver figura 5):

Fig. 45: Aspecto del logo antes y después de aplicar el filtro SVG.

Otro tipo de filtro interesante es tiene que ver con el concepto de iluminación. Existen dos primitivas de iluminación: ***feDiffuseLighting*** y

feSpecularLighting. Y además disponemos de 3 tipos de iluminaciones distintas: luz distante (***feDistantLight***), luz puntual (***fePointLight***) y luz de foco (***feSpotLight***).

Igualmente, disponemos de filtros compuestos (***fecomposite***), filtros de fusión (***femerge***), que permiten aplicar más de un filtro simultáneamente, efectos de distorsión de imagen (***fegaussianblur***), y muchos otros, que pueden verse en la dirección indicada más arriba.

A continuación incluimos un ejemplo mas completo que aplica un conjunto de filtros a las 3 figuras geométricas definidas más arriba:

```
<defs>

<filter id="Filtro">
    <fegaussianblur in="SourceAlpha" stddeviation="4"
result="blur" />
    <feoffset in="blur" dx="4" dy="4" result="offsetBlur"
/>
    <fespecularlighting in="blur" surfacescale="5"
specularconstant="1"
        specularexponent="10" lighting-color="white"
result="specOut">
        <fepointlight x="-5000" y="-10000" z="20000" />
    </fespecularlighting>
    <fecomposite in="specOut" in2="SourceAlpha"
operator="in" result="specOut" />
    <fecomposite in="SourceGraphic" in2="specOut"
operator="arithmetic"
        k1="0" k2="1" k3="1" k4="0" result="litPaint" />
    <femerge>
        <femergenode in="offsetBlur" />
        <femergenode in="litPaint" />
    </femerge>
</filter>

<g id="CirRectTriangulo">
   <g>
     <circle cx="185" cy="80" r="140" fill="yellow" />
      <g>
```

```
        <rect x="145" y="45" width="190" height="170"
fill="red" />
      </g>
       <path d="M360,40 L435,80 L360,120 z"
fill="#4444ff" />
    </g>
</g>
</defs>

<g filter="url(#Filtro)">
     <use x="150" y="50" xlink:href="#CirRectTriangulo
"/>
</g>
```

Como podemos ver, primero se establecen las definiciones, contenidas en un elemento ***<defs>***, y agrupadas por dos criterios: las puramente gráficas, y las que tienen que ver con los efectos a aplicar.

Una vez hecho esto, un nuevo elemento ***<g>***, utiliza la etiqueta ***<use>*** para hacer referencia a los elementos gráficos (identificador ***#CirRectTriangulo***) y aplicarles el filtro (***#Filtro***) correspondiente asignando el atributo ***filter***.

La salida gráfica corresponde a la de la figura:

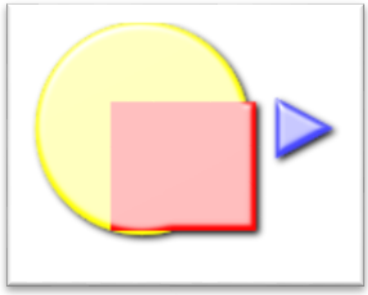


Fig. 46: Salida del código anterior.

Donde podemos apreciar los efectos de iluminación, sombra y biselado, que se han definido al principio.

Para más información, puede ver las referencias, al final del capítulo.

El modelo de contenido

El modelo de contenido es lo que define qué elementos pueden estar anidados —están permitido como descendientes- en un determinado elemento.

En general, HTML4 tenía dos categorías principales de elementos, "***inline***" (e.g. ***span***, ***img***, texto) y "nivel de bloque" (***div***, ***hr***, ***table***). Y algunos elementos no adecuaban a ninguna categoría.

De esta forma, algunos elementos permitían elementos anidados "***inline***" (como ***p***), otros permitían elementos de "nivel de bloque" (como ***body***), y otros ambos (como ***div***), mientras que había elementos que no permitían más que otros elementos específicos (por ejemplo ***dl*** o ***table***) o no permitían hijos en absoluto (como ***link***, ***img*** o ***hr***).

Y hay que advertir la diferencia entre que un elemento sí mismo esté en una categoría determinada, y que permita tener unos contenidos de otra categoría; por ejemplo, el elemento de p es en sí mismo un elemento de "nivel de bloque", pero tiene un modelo de contenido "***inline***".

Para mayor confusión, HTML4 tenía diferentes reglas del modelo de contenido dependiendo de sus *sabores Strict*, *Transitional* y *Frameset* (que se definían en el DOCTYPE). Y para hacerlo más confuso aún, CSS utiliza el término "elemento de nivel de bloque" y "elemento de nivel de línea" para su modelo de formato visual, que se relaciona de propiedad ***display*** de CSS y no tiene nada que ver con las reglas del modelo de contenido de HTML.

HTML no utiliza los términos "*nivel de bloque*" o "*en línea*" como parte de las reglas de su modelo de contenido, para reducir la confusión con CSS. Sin embargo, tiene más categorías de HTML4, y un elemento puede ser parte de ninguno de ellos, uno de ellos, o varios de ellos. Así pues podemos tener:

- Contenido de metadatos, por ejemplo ***link***, ***script***.

- Contenido dinámico (*Flow content*), como ***span***, ***div*** o texto. Esto es aproximadamente como "*block level*" e "*inline*" de HTML4 juntos.
- Secciones de contenido, por ejemplo, ***aside***, ***section***.
- Contenido de cabecera, como por ejemplo, ***h1***.
- Contenido textual en forma de frases como en span, img o texto. Esto es más o menos como "*en línea*" de HTML4. Los elementos que son de fraseo también son contenido dinámico.
- Contenido incrustado, como, por ejemplo, ***img***, ***iframe*** o ***svg***.
- Contenidos interactivos, por ejemplo a ***button*** o ***label***. El contenido interactivo no puede anidarse.

Como un amplio cambio desde HTML4, HTML ya no tiene ningún elemento que sólo acepte lo que HTML4 llamaba " elementos de nivel de bloque"; por ejemplo el elemento ***body*** permite ahora contenido dinámico. Por lo tanto, esto está más cerca HTML4 *Transitional* que del HTML4 *Sctrict*.

Otros cambios incluyen:

- El elemento de ***address*** ahora permite flujo de contenido, pero con ningún descendiente contenido rumbo, sin descendientes contenidos seccionamiento y no ***header***, ***footer*** o descendientes de elemento de ***address*** .
- HTML4 permitía ***object*** dentro de ***head***. HTML no lo hace.
- WHATWG HTML permite que ***link*** y ***meta*** sean descendientes de ***body*** si utilizan atributos de *microdata*.
- El elemento ***noscript*** era un elemento de "nivel de bloque" en HTML4, pero es de fraseo contenido en HTML.

- Los elementos de ***table***, ***thead***, ***tbody***, ***tfoot***, ***tr***, ***ol***, ***ul*** y ***dl*** pueden estar vacíos en HTML.
- Los elementos de ***table*** tienen que estar conformes con el modelo de tabla (por ejemplo dos celdas no pueden superponerse).
- El elemento ***table*** ahora no permite elementos secundarios directos tipo ***col***. Sin embargo, el analizador HTML integra implícitamente un elemento ***colgroup***, así que este cambio no debería afectar el contenido tipo *text/html*.
- El elemento ***table*** permite ahora el elemento ***tfoot*** sea el último hijo.
- El elemento ***caption*** permite contenido dinámico, pero sin elementos descendientes.
- El elemento ***th*** ahora permite contenido dinámico, pero sin descendientes ***header***, ***footer***, o contenido de sección, o cabeceras.
- ***a*** tiene ahora un modelo de contenido transparente (excepto que no permite contenidos interactivos en los descendientes), lo que significa que tiene el mismo modelo de contenido que su padre. Esto significa que ***a*** puede contener ahora elementos ***div,*** por ejemplo, si sus padres permiten contenido dinámico.
- Los elementos ***ins*** y ***del*** también tienen un modelo de contenido transparente. HTML4 tenía reglas similares que no podían ser expresadas en la DTD (*Document Type Definition*).
- El elemento de ***object*** también tiene un modelo de contenido transparente, después de sus descendientes tipo ***param***.
- El elemento ***map*** también tiene un modelo de contenido transparente. El elemento ***area*** se considera como contenido de

fraseo si hay un ancestro del elemento ***map***, lo que significa que no necesitan ser hijos directos de ***map***.

- El elemento ***fieldset*** ya no requiere un descendiente tipo ***legend***.

Conclusión

Hay muchos cambios interesantes en las nuevas etiquetas disponibles en HTML 5. El nivel de disponibilidad de las nuevas etiquetas en los navegadores es ahora bastante alto, y además, contamos con librerías especializadas para proveer de mecanismos de "*fallback*", y aprovechar al máximo su potencial.

Tenemos la capacidad de conseguir una mayor visibilidad, al tiempo que se acerca el sueño de codificar una vez para cualquier dispositivo o plataforma. Pero las etiquetas no lo son todo. Los cambios en los atributos han supuesto un complemento fundamental para la construcción de la nueva Web. Ese es el objetivo del capítulo siguiente.

Referencias

- "*HTML 5. The missing Manual*", **Mathew McDonald**, O'Reilly, 2011
- "*HTML 5 Step by Step*", **Faithe Wempen**, Microsoft Press, 2011
- "*Introducing HTML 5. 2nd Edition*", **Bruce Lawson** y ***Remy Sharp***, Riders, 2012
- "*HTML5 Mastery: Semantics, Standards, and Styling*", Anselm Bradford y Paul Haine, Ed. Friends of Ed. 2011
- "*HTML 5. A technical specification for Web developers*". Versión específica para desarrolladores del WHATWG: http://developers.whatwg.org/
- *"How To Edit YouTube Videos and Add Video Annotations"*: http://www.htmlgoodies.com/beyond/video/how-to-edit-youtube-videos-and-add-video-annotations.html

- "*HTML5 Canvas: Native Interactivity and Animation for the Web*", **Steve Fulton** y **Jeff Fulton**. O'Reilly Media, 2011
- Estándar SVG: http://www.w3.org/TR/SVG11/
- Tutorial de SVG: http://www.w3.org/Consortium/Offices/Presentations/SVG/0.svg

Capítulo 03 | Los Atributos en HTML 5

El documento oficial de diferencias entre el estándar actual y el anterior, dedica buena parte de su contenido a los atributos que acompañan a los nuevos elementos, a los nuevos atributos presentes en elementos antiguos, y al conjunto de atributos que se consideran obsoletos.

Entre las novedades, cabe destacar conjuntos completos de propiedades, como el conjunto ***aria-****, cuyo objetivo es dar la cobertura adecuada a las necesidades que desde hace tiempo se echaban en falta en accesibilidad, o la o la capacidad de crear atributos personalizados, que está siendo usada profusamente en las aplicaciones "Tienda de Windows" en Windows 8.

Nuevos atributos

Varios atributos se han introducido en diversos elementos que ya formaban parte de HTML4:

- Los elementos ***a*** y ***area*** tienen el nuevo atributo ***download*** en los documentos de WHATWG HTML y W3C HTML5.1. WHATWG HTML también tiene el atributo ***ping***.
- El elemento ***area***, por coherencia con los elementos ***a*** y ***link***, ahora también tiene los atributos la ***hreflang*** , ***type*** y ***rel***.
- El elemento ***base*** puede tener ahora un atributo ***target***, sobre todo por consistencia con el elemento ***a***. (Esto ya está ampliamente soportado).
- El elemento ***meta*** tiene ahora un atributo ***charset***, también ampliamente soportado y proporciona una buena manera de especificar la codificación de caracteres del documento. Tiene sentido para ayudar en la migración de documentos escritos en XHTML.
- En WHATWG HTML y W3C HTML5.1, el elemento ***table*** tiene ahora un atributo ***sortable*** y el elemento ***th*** tiene un atributo ***sorted***, que proporcionan un medio para ordenar las columnas de la tabla.
- El elemento ***fieldset*** permite ahora el atributo ***disabled*** que desactiva todos los controles descendientes (excluyendo aquellos que son descendientes del elemento ***legend***) y el atributo ***name***, que se puede utilizar para el acceso mediante *scripts*.
- El elemento ***textarea*** tiene dos nuevos atributos, ***maxlength*** y ***wrap*** que controlan la longitud máxima de entrada y el ajuste de línea, respectivamente.
- El elemento ***form***, tiene un atributo ***novalidate*** que permite desactivar la validación del formulario.
- Tanto ***input*** como ***button*** tienen ahora un conjunto de atributos ***formaction***, ***formenctype***, ***formmethod, formnovalidate*** y ***formtarget***, que permiten anular el comportamiento predeterminado de sus propiedades correspondientes.

- El elemento ***menu*** tiene dos nuevos atributos: ***type*** y ***label***. Permiten que el elemento se transforme en un menú que se encuentra en las interfaces de usuario típicos, así como proporcionar menús de contexto.
- En WHATWG HTML y W3C HTML5.1, el elemento ***button*** tiene un nuevo atributo ***menu***, que se utiliza junto con los menús emergentes.
- El elemento ***style*** tiene un nuevo atributo ***scope*** que puede ser utilizado para proporcionar un ámbito de ejecución a las hojas de estilo. Las reglas de un estilo con ese atributo, solo se aplican al árbol local.
- El elemento ***script*** tiene un nuevo atributo llamado ***async*** que influye en la carga y ejecución del código.
- El elemento ***HTML*** tiene un nuevo atributo llamado ***manifest*** que apunta a un manifiesto de caché de la aplicación, y se utiliza junto con la API para aplicaciones web *offline*.
- El elemento ***link*** tiene un nuevo atributo llamado ***sizes***. Puede ser utilizado en conjunción con el icono para indicar el tamaño del icono de referencia, permitiendo de este modo iconos de dimensiones distintas.
- El elemento ***ol*** tiene un nuevo atributo llamado ***reversed***. Cuando está presente, indica que el orden de la lista es descendente.
- El elemento ***iframe*** tiene ahora tres nuevos atributos llamados ***sandbox***, ***seamless*** y ***srcdoc*** que permiten contenido seguro (***sandboxing***), por ejemplo, los comentarios del blog.
- El elemento ***object*** tiene un nuevo atributo llamado ***typemustmatch*** que permite incrustar recursos externos de forma más segura.
- El elemento ***img*** tiene un nuevo atributo llamado ***crossorigin*** para permitir el uso de CORS en una petición y, si esta es correcta, permite que los datos de la imagen sean leídos con el API ***canvas***. En WHATWG HTML y W3C HTML5.1, el elemento ***script*** tiene un atributo ***crossorigin*** para permitir que los errores en un *script* que deben notificarse al evento ***onerror*** contengan información sobre

el error. WHATWG HTML y W3C HTML5.1 también contienen un atributo ***crossorigin*** en el elemento ***link***.

- En WHATWG HTML, el elemento ***img*** tiene un nuevo atributo llamado ***srcset*** para soportar múltiples imágenes de diferentes resoluciones y diferentes imágenes de diferentes tamaños de ***viewport*** (ventana gráfica).

Vamos a analizar con más detalle aquellos que tienen una mayor repercusión para los desarrolladores y autores de sitios/aplicaciones web.

Nuevos atributos y semántica del elemento *img*

El elemento ***img*** es reconocido por la W3C como uno de los más importantes de una página o aplicación Web. De ahí que se han establecido con todo tipo de detalle los valores que puede recibir cada uno de sus atributos e incluso los significados que tienen las diversas combinaciones de atributos presentes y ausentes de un elemento de este tipo[37].

Ahora disponemos de un atributo ***srcset***. Si está presente, su valor debe consistir en una o varias cadenas de candidato de imagen, cada uno separado del siguiente por un carácter coma (,). Este atributo permite a los autores ofrecer imágenes alternativas para entornos con pequeñas pantallas o pantallas con mayor densidad de píxeles (esto está en fase de implantación en los navegadores).

```
<img alt="The Breakfast Combo"
     src="banner.jpeg"
     srcset="banner-HD.jpeg 2x, banner-phone.jpeg 100w,
banner-phone-HD.jpeg 100w 2x">
```

[37] Para un estudio al detalle, recomendamos la página oficial http://www.whatwg.org/specs/web-apps/current-work/multipage/embedded-content-1.html#attr-img-crossorigin

El atributo ***crossorigin*** es una configuración CORS. Su propósito es permitir imágenes de sitios de terceros que permiten el acceso cruzado a esas fuentes para usarse con el elemento ***canvas***.

Estados de *img*

Un elemento ***img*** está siempre en uno de los siguientes estados:

- **Original**: El agente de usuario no ha obtenido los datos de imagen.
- **Parcialmente disponible**: El agente de usuario ha obtenido algunos de los datos de imagen.
- **Totalmente disponible**: El agente de usuario ha obtenido todos los datos de la imagen y por lo menos están disponibles sus dimensiones.
- **Roto**: El agente de usuario ha obtenido todos los datos de imagen que se pueden obtener, pero aún no puede decodificar la imagen lo suficiente para obtener las dimensiones de la imagen (por ejemplo, la imagen está dañada, el formato no es compatible, etc.).

Cuando un elemento ***img*** está disponible (total o parcialmente), proporciona una fuente gráfica cuyo ancho es el ancho intrínseco de la imagen, cuya altura es la altura intrínseca de la imagen, y cuya relación de aspecto es la de la imagen (puede, por tanto, ser manipulada mediante *scripting*).

Significado según la presencia de atributos

Lo que representa un elemento ***img*** para el motor de interpretación, depende de los atributos ***src*** y ***alt***.

- Si se establece el atributo ***src*** y el atributo ***alt*** se establece en la cadena vacía:

    - La imagen es decorativa o suplementaria para el resto de los contenidos, y posiblemente redundante con otra información en el documento.

    - Si la imagen está disponible y el agente de usuario está configurado para mostrar esa imagen, entonces el elemento representa datos relativos a la imagen del elemento.
    - De lo contrario, el elemento no representa nada, y puede ser omitido totalmente de la presentación. Los agentes de usuario pueden proporcionar a éste una notificación de que una imagen está presente pero se ha omitido su visualización.

- Si se establece el atributo ***src*** y el atributo ***alt*** se establece en un valor que no esté vacío

    - La imagen es una parte clave del contenido; el atributo ***alt*** da un equivalente textual o reemplazo de la imagen.
    - Si la imagen está disponible y el agente de usuario está configurado para mostrar esa imagen, entonces el elemento representa datos de imagen del elemento.
    - De lo contrario, el elemento representa el texto dado por el atributo ***alt***.

- Si se establece el atributo ***src*** pero NO el atributo ***alt***

    - La imagen puede ser una parte clave del contenido, pero no hay ningún equivalente textual de la imagen disponible.
    - Nota: En un documento conforme, la ausencia del atributo ***alt*** indica que la imagen es una parte clave del contenido, pero que no estaba disponible un reemplazo textual de la imagen cuando se generó ésta.

El documento oficial del estándar brinda algunos ejemplos clarificadores de esta situación con varias propuestas sencillas, indicando su significado, que reproducimos literalmente a continuación. En cada uno de los siguientes casos, se utiliza la misma imagen, pero el texto ***alt*** es

diferente cada vez. El lector apreciará en los ejemplos que -con estas combinaciones- se busca aportar contenido de tipo semántico:

La imagen es del municipio ***Carouge*** en el cantón de Ginebra en Suiza:

Aquí se utiliza como un icono suplementario:

```
    <p>I lived in <img src="carouge.svg" alt="">
Carouge.</p>
```

Aquí se utiliza como un icono que representa la ciudad:

```
    <p>Home town: <img src="carouge.svg"
alt="Carouge"></p>
```

Aquí se utiliza como parte de un texto sobre la ciudad:

```
    <p>Carouge has a coat of arms.</p>
    <p><img src="carouge.svg" alt="The coat of arms
depicts a lion, sitting in front of a tree."></p>
    <p>It is used as decoration all over the town.</p>
```

Aquí se utiliza como una manera de apoyar un texto similar donde se aporta una descripción, en lugar de como una alternativa a la imagen:

```
    <p>Carouge has a coat of arms.</p>
    <p><img src="carouge.svg" alt=""></p>
    <p>
The coat of arms depicts a lion, sitting in front of a
tree. It is used as decoration all over the town.
    </p>
```

Aquí se utiliza como parte de una historia:

```
<p>He picked up the folder and a piece of paper fell
out.</p>
    <p>
        <img src="carouge.svg" alt="Shaped like a shield,
the paper had a red background, a green tree, and a
```

```
yellow lion with its tongue hanging out and whose tail
was shaped like an S.">
    </p>
<p>He stared at the folder. S! How had he not seen that
before? It all came together now. The phone call where
Hector had referred to a lion's tail, the time Marco had
stuck his tongue out...
    </p>
```

Finalmente, aquí, en el momento de la publicación no se sabe la imagen que será, sólo que será un escudo de algún tipo y, por lo tanto, no puede proporcionarse ningún texto de reemplazo. En cambio, se proporciona sólo un título breve para la imagen, en el atributo ***title***:

```
<p>The last user to have uploaded a coat of arms uploaded
this one:</p>
      <p><img src="last-uploaded-coat-of-arms.cgi"
title="User-uploaded coat of arms."></p>
```

Observe como el atributo ***src*** apunta a un elemento ***cgi*** en vez de a una imagen.

Descarga diferida de recursos

En algunos casos, los recursos están diseñados para uso posterior en lugar de su visualización inmediata. Para indicar que un recurso está destinado a ser descargado para ser usado más adelante, un atributo ***download*** en un elemento ***a*** o ***area*** permite crear el hipervínculo a ese recurso.

El atributo puede añadir además un valor para especificar el nombre del archivo que los agentes de usuario deben usar cuando se guarde el recurso en un sistema de archivos.

Los atributos de <*input*>

A todas estas novedades, hay que añadirle las que se vinculan con el

elemento de entrada ***input,*** que es de los que más modificaciones han sufrido por la necesidad de adaptar los mecanismos de entrada de datos del usuario a las interfaces de hoy, tanto en páginas Web como de las aplicaciones.

En total, el conjunto de valores posibles para el actual atributo ***type*** y sus significados, es el siguiente:

- **tel** (teléfono)
- **search** (texto a buscar)
- **url** (dirección de Internet)
- **email** (dirección de correo)
- **datetime** (fecha y hora)
- **date** (fecha)
- **month** (mes)
- **week** (semana)
- **time** (hora)
- **datetime-local** (fecha/hora en formato local)
- **number** (número)
- **range** (rango de valores)
- **color** (color)

La mayor parte tienen un significado semántico muy claro (que limita o indica el tipo de dato a introducir), y están pensados para permitir un mayor nivel de conformidad respecto al tipo de entradas de datos y un alto nivel de validaciones cuando este globalmente implementado.

Aparte de la semántica y las validaciones, también introduce cambios visuales, ya que permite, por ejemplo, que se asocien nuevos controles de interfaz de usuario con algunos de estos atributos, como puede ser el caso de los calendarios para la introducción de fechas, o controles del tipo ***numericUpDown***, para datos numéricos.

Recordemos que la etiqueta ***<input>*** -por definición- debe de ubicarse en un formulario (etiqueta ***<form>***), así que –en realidad- estos cambios y otros de corte similar, forman parte de una API más global asociada con este elemento. No obstante, hablaremos de la etiqueta ***<form>*** con más detalle más adelante en este capítulo.

Comportamiento interno y reflejo en el DOM

Internamente, cuando se define una etiqueta con uno de los nuevos atributos, el elemento queda marcado semánticamente de cara al Modelo de Objetos de Documento (DOM), de forma que los buscadores puedan referenciar los valores adecuados.

Estos atributos se complementan muchas veces con otros que complementan su funcionalidad, como ***required, contextmenu, data-, draggable, max, min*** o ***step***, sio bien no todos ellos tienen sentido para cualquier valor de ***type***.

En la terminología propia del sitio global de documentación para desarrolladores de Microsoft, se dice que cada uno de estos atributos define, por tanto, un **estado** del elemento **<input>**[38].

<*input*> y los formatos de entrada

En cuanto al soporte, hay que distinguir entre el funcional y el visual. Los navegadores actuales soportan la funcionalidad respecto al DOM, y permiten una programación asociada con cada estado. En algunos casos, hay además cambios visuales, que –según el estándar- son totalmente a gusto de los agentes de usuario.

Así que, disponemos de varias posturas en la implementación. Desde la de Opera, que implementa elementos de tipo visual ya definidos, y que incluyen hasta los mensajes predeterminados (cuando son necesarios), hasta la implementación actual de IE10 o Chrome, que optan por mínimos componentes visuales, prefiriendo que sea el usuario quien configure ese aspecto, o implementando solo algunos casos evidentes.

La descripción general de los atributos disponibles es la siguiente:

- Atributo ***tel***: No adopta ningún aspecto visual especial. Si se requiere un formato, puede usarse el atributo ***pattern***. (lo vemos más adelante)

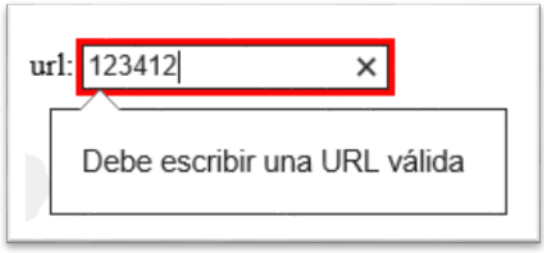
[38] http://msdn.microsoft.com/es-es/library/hh673544(v=vs.85)

- Atributo ***search***: Idéntico al ***text***, pero marcado semánticamente.

El siguiente código muestra la sintaxis de estos atributos, que no aportan ninguna diferencia visual especial respecto al tipo ***text*** en ningún navegador:

```
<label for="tel">tel: <input type="tel" id="tel"/>
</label>
<label for="search">search: <input type="search"
id="search" /></label>
```

- Atributos ***url*** y ***mail***: Marcados semánticamente, también controlan el contenido, de forma que no admiten direcciones o correos sintácticamente inválidos cuando se produce el evento de validación interno y se envía el contenido del formulario al servidor. Visualmente, generan los siguientes mensajes de error (junto con un aspa que permite vaciar su contenido (Fig. 1 y 2):



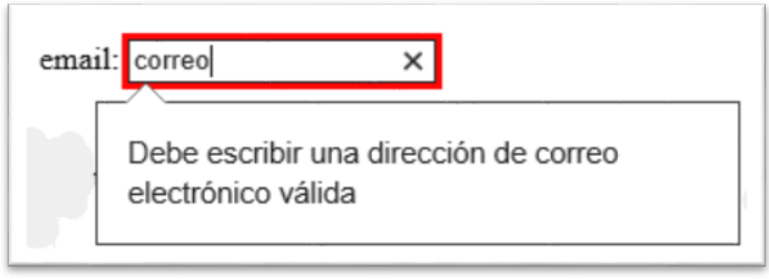


Fig. 1 y 2: Salida en IE10 con un intento de enviar datos incorrectos en url/email

- Atributos relacionados con el tiempo: ***date, datetime, datetime-local, month, week*** y ***time***: Permiten una especificación semántica muy detallada al expresar conceptos relacionados con el tiempo. La parte visual, probablemente, acabará por ser implementada según el dispositivo de visualización y el agente de usuario encargado de la tarea. De momento, Opera ofrece la salida que vemos en la figura 3, si usamos el siguiente código fuente (formateado con un estilo, para su separación en líneas distintas).

```
<label for="email">email: <input type="email" id="email"
/></label>
<label for="datetime">datetime: <input type="datetime"
id="datetime" /></label>
<label for="datetime-local">datetime-local: <input
type="datetime-local" id="datetime-local" /></label>
<label for="date">date: <input type="date" id="date"
/></label>
<label for="time">time: <input type="time" id="time"
/></label>
<label for="month">month: <input type="month" id="month"
/></label>
<label for="week">week: <input type="week" id="week"
/></label>
```

Utilizamos etiquetas ***<label>*** debido a las numerosas ventajas que tienen desde el punto de vista de la accesibilidad. El código de formato (el estilo) utiliza el siguiente código fuente (veremos este apartado en el siguiente capítulo)

```
<style>
    label:before { content: '\a\a\a'; white-space: pre;
    }
</style>
```

Ese estilo produce 3 retornos de carro y un tratamiento especial de los espacios en blanco.

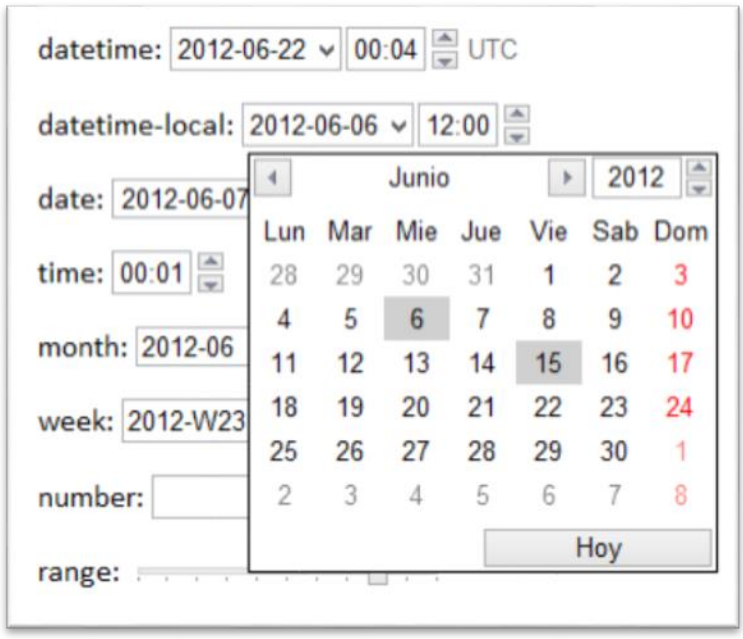


Fig. 3: Salida del código anterior y de la etiqueta <input type="datetime-local"> en Opera

Como hemos podido ver, existen distintos atributos especializados en la introducción de fechas/horas, y varios formatos. Como siempre, la especificación no indica el aspecto visual que debe de adoptar el control, pero se espera que la mayoría de los navegadores implemente algún tipo de selector, como el que hemos visto en el ejemplo inicial con Opera 12.

Por poner un ejemplo, en el caso de los meses (***type="month"***), el navegador debe guardar internamente un valor entre 1 y 12, aunque puede para ello presentar un calendario completo donde la selección se limite a calcular el valor del mes a partir de la fecha seleccionada por el usuario.

Aparte de ello, y, aunque no se trata de números sino de otro tipo de datos, la especificación dice que los tipos de fecha deben incluir los atributos adicionales ***value, max*** y ***min*** que ya hemos visto a propósito de los números, y así poder indicar el valor actual y los rangos permitidos de fechas, meses, semanas, etc.

- Atributos para valores escalares: ***number, range*** y ***color***: permiten la introducción de números en distintos formatos: numérico con variantes, rango de valores y colores. Su sintaxis es muy similar a los anteriores con un aspecto como el siguiente sin utilizar ningún atributo de configuración:

```
<label for="number">number: <input type="number"
id="number" /></label>
<label for="range">range: <input type="range" id="range"
/></label>
<label for="color">color: <input type="color" id="color"
/></label>
```

El primero, tiene un comportamiento similar a un elemento de tipo ***numericUpDown*n.** En Opera, incluso podemos verlo con ese mismo aspecto. Dispone de los atributos ***max***, ***min*** y ***step***, que condicionan el tipo de entradas al rango indicado por los dos primeros y, si el tercero está presente, a los valores múltiplo intermedios que puedan adoptar, como vemos en los dos mensajes de error que muestra IE10 para el siguiente código fuente:

```
<label for="number">number: <input type="number"
id="number" max="100" min="0" step="5" /></label>
```

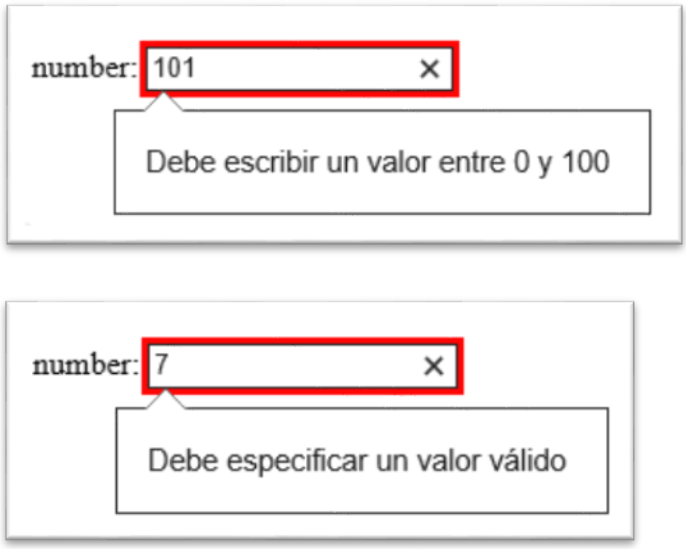


Fig. 4: Salida en IE10/11 del código anterior, mostrando los mensajes de error predeterminados

Lógicamente, en el primer caso estamos fuera de rango, y en el segundo, el valor no es múltiplo de 5 (condición ***step***).

En el segundo caso, la funcionalidad es clara y similar a la de ***number*** (cuando se han establecido los mismos parámetros de delimitación y avance), pero la diferencia es que, los navegadores muestran una interfaz visual bien definida.

En el caso de IE10/11, si utilizamos el siguiente código fuente:

```
<label for="range">range: <input type="range" id="range"
/></label>
```

Obtendremos un *slider* que ya coincide con el estilo visual propuesto por las aplicaciones Metro Style de Windows 8:

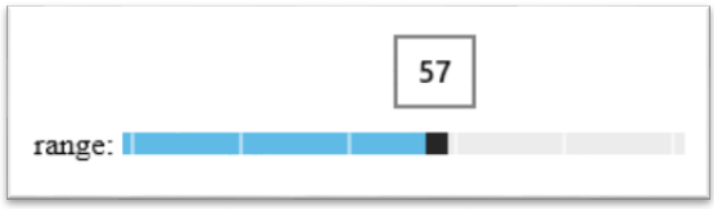


Fig. 5: Salida del elemento range en ejecución visto en IE10

Otros atributos permiten acotar aún más las opciones de entrada, facilitando la labor de validación y minimizando errores. Es el caso de ***maxlength*** y ***minlength*** (longitud máxima y mínima), para las cajas de texto.

Finalmente, el atributo ***color*** permite la introducción de un dato de color válido, aunque la implementación visual solo está completa en Opera en el momento de escribir estas líneas, pero parece que la versión final de IE10 ofrecerá una interfaz similar a la que ofrece la salida que vemos en la figura 6:

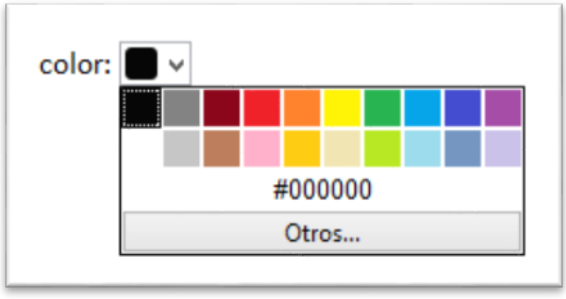


Fig.6: Salida del atributo color en Opera con el código anterior

De hecho, esta opción ya está disponible en V. Studio 2012, como indicábamos en el capítulo 2 dedicado a las herramientas.

Una vez seleccionado un valor, el atributo ***value*** del elemento guarda una representación de cadena válida para ser interpretada como un color por los agentes de usuario.

No obstante, la programación para formularios en HTML 5 no concluye ahí. La especificación no indica la manera en que cada fabricante

implemente lo exigido en el documento formal, con tal de que lo haga. Esto supone, por un lado, una gran capacidad de elección, y permite a los navegadores aprovechar la tecnología que ya soporten para implementar el estándar, y por otro, podemos imaginar que –mientras lo indicado en la especificación se cumpla- cada navegador tratará de aportar aquellos elementos que vayan más en consonancia con las necesidades de sus usuarios.

En cualquier caso, los navegadores antiguos no generarán fallos, sino que, simplemente, mostrarán la caja de texto habitual, sin elementos añadidos.

Atributos complementarios de control y visuales

Al usar la etiqueta ***<input>*** en Visual Studio 2012, también apreciaremos cómo el editor sugiere otros atributos complementarios, que ayudan a configurar una experiencia de entrada de datos más consistente. Los revisamos a continuación.

El atributo placeholder

Un marcador de posición en HTML5 funciona en forma similar a como estamos acostumbrados en las aplicaciones de escritorio. Se mostrará un texto al usuario en un tono más apagado del normal, indicándole qué es lo que se espera que introduzca en la caja correspondiente.

Se implementa como un atributo al que se asigna la cadena a mostrar y es común a casi todos los valores nuevos del atributo *type*.

Basta con indicar el texto deseado y podremos verlo en la salida en pantalla como aparece en la Fig. 7. Si el usuario pulsa dentro de la caja de texto, el texto apagado desaparece, permitiendo la edición, y –caso de haberlo dejado en blanco- vuelve a aparecer al perder el foco. El siguiente código genera la salida indicada:

```
<input type="tel" placeholder="formato: (xxx) xx-xx-xxx" />
```

Fig. 7: Placeholder en funcionamiento con un atributo tel, visto en IE10/11

También es posible formatear la salida utilizando una pseudo-clase de CSS denominada ***–ms-input-placeholder*** (veremos CSS con más profundidad en el siguiente capítulo). Se utiliza como cualquier otra directiva CSS y permite que el analizador busque las apariciones de cualquier elemento que contenga un atributo ***placeholder*** y lo presente visualmente como le indiquemos.

Por ejemplo, si añadimos el código siguiente en la página anterior:

```
/* Estilo para placeholder solamente */
input.address:-ms-input-placeholder
{
    font-variant:small-caps ;
    background-color: #ff6a00;
    color: Navy;
}
```

Y, si asociamos el atributo ***class*** a *marcador* en un elemento que contiene un atributo ***placeholder***:

```
<label for="tel">tel: <input type="tel" id="tel"
placeholder="formato: (xxx) xx-xx-xxx" class="marcador"
/> </label>
```

Obtendremos la siguiente salida en IE10:

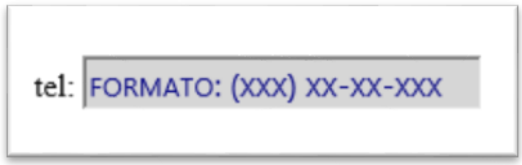


Fig. 8: Salida del código anterior aplicando el nuevo formato

El atributo *pattern*

<placeholder> suele implementarse en muchos casos en conjunción con el atributo ***pattern***, que permite configurar una expresión regular para controlar la validación del contenido. La idea es que, además de la sugerencia de formato, el programador pueda implementar un patrón de validación contra el que poder comprobar el texto introducido. El tipo del elemento ***<input>*** debe ser ***email, password, tel, text, search*** o ***url.***

Dado que las expresiones regulares son complejas de escribir debido a su sintaxis, recomendamos al lector que utilice uno de los muchos generadores de este tipo que existen en la red, o que instale uno de los que están disponibles directamente como complementos si está utilizando Visual Studio[39].

Por poner uno de los casos más simples, si tuviéramos que restringir la entrada de datos en una caja de texto, y quisiéramos un primer nivel de validación garantizando que se introducen solamente números y un máximo de 3 dígitos, como podía ser en el caso de la edad, podríamos utilizar el código siguiente:

```
<label for="Edad">
<input type="text" id="Edad" title="Utilice solamente
números" pattern="\d{3}" placeholder="Introduzca hasta 3
dígitos para la edad" required="required" />
</label>
```

[39] Para un listado de expresiones comunes de este tipo, ver "*Using Regular Expressions in Visual Studio*", en http://msdn.microsoft.com/en-us/library/2k3te2cs(v=vs.110).aspx.

En caso de que el dato introducido no corresponda con la evaluación de la expresión regular (que la realiza el motor de interpretación al validar el contenido del formulario), el navegador puede mostrar una etiqueta flotante similar a la anterior o podemos controlar la situación mediante su API de validación que veremos más adelante. Por ejemplo, si en ese mismo código cometemos un error, el navegador IE10 nos mostrará el mensaje que vemos en la figura 9:

Fig. 9: Mensaje de error en IE10/11 vinculado a una expresión regular.

El atributo *autofocus*

Otra característica muy solicitada era la de poder decidir desde el diseño cuál de los controles de un formulario debe tomar el foco cuando se carga la página. Para ello se utiliza uno de los atributos de "presencia", esto es, atributos que indican un valor verdadero asignado a su propio nombre.

Así pues, si añadimos el siguiente código al anterior, y queremos que automáticamente tome el foco el tercer ***<input>***, bastaría con lo siguiente:

```
<input id="Segundo" />
<input id="Tercero" autofocus="autofocus" />
```

(La salida se la puede imaginar el lector).

El atributo *autocomplete*

La característica de autocompletar (la posibilidad de que el navegador

"recuerde" una entrada de datos de uso habitual), ya la implementaban la mayoría de los navegadores y es ahora parte del estándar. Su sintaxis permite la asignación de dos valores (***on***/***off***), activándola o no, según nos encontremos en un campo de seguridad restringida.

Sintácticamente, podemos expresar que no queremos que un campo sea recordado con algo tan simple como:

```
<input type="text" id="NumSegSocial" autocomplete="off"/>
```

La razón es que, habitualmente, este atributo estará marcado como "*on*" a nivel de formulario, pudiendo hacer excepciones a ese comportamiento global mediante esta opción.

> **Nota**: Téngase en cuenta que esta característica viene deshabilitada en algunos navegadores de forma predeterminada. Ese es el caso de IE10/11, así que tendremos que acceder a la configuración de "Opciones de Internet" y en la solapa "Contenido", habilitar esta opción (Ver figura 10).

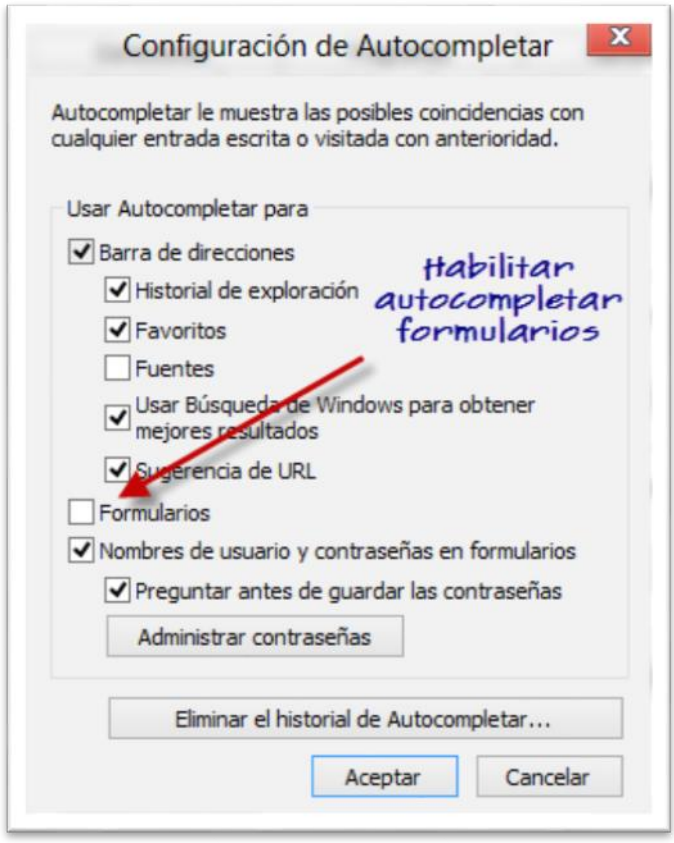


Fig. 10: Configuración de Autocompletar en el navegador IE10

El lector pudiera tener que configurar esta opción manualmente para los distintos navegadores que utilice, ya que los valores predeterminados no tienen por qué coincidir entre los distintos agentes de usuario y versiones instaladas.

El atributo *list* de <input> y el elemento <datalist>

El elemento **<datalist>** no tiene definida ningún tipo de presentación, pero cuando se utiliza conjuntamente con otro control, puede presentar un cuadro combinado (*ComboBox*), que permite al usuario la selección de una entre varias opciones posibles en modo de despliegue (facilita la

implementación en HTML de valores enumerados). Se complementa con elementos **<option>** para indicar cada uno de los posibles valores. Típicamente, sirve como repositorio de datos para ser utilizado con un elemento **<input type = {***cualquier de los nuevos atributos***} />**.

Para ello, ***<input>*** dispone ahora de un nuevo atributo ***list***, con el que podemos enlazar el elemento para mostrar la lista de opciones.

Por ejemplo, podemos combinar un elemento **<datalist>** con una entrada **<input>** de tipo email, con el siguiente código fuente:

```
<datalist id="contactos">
   <option value="info@danysoft.com" label=" Información
en Danysoft" />
   <option value="admin@danysoft.com" label="
Administración de Danysoft" />
</datalist>
<input type="email" id="emailDanysoft" list="contactos"
/>
```

Lo que generaría una salida de una caja de texto estándar, pero, cuando pulsamos sobre ella, se despliegan los valores asignados, tal y como vemos en la Fig. 11.

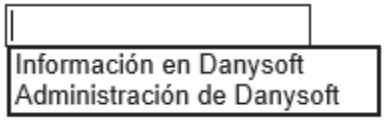


Fig. 11: DataList utilizado en combinación con un elemento <input type="email" /> visto en IE10

También podemos conseguir una funcionalidad equivalente con la antigua (HTML4) etiqueta **<optgroup>.** Esta etiqueta admite un atributo ***label***, y si la utilizamos con un elemento **<select>**, podemos ordenar los ítems por categorías y obtener un mayor nivel de personalización y de agrupación. Como vemos, en muchos casos se trata simplemente de otras alternativas.

Basándonos en lo anterior, podríamos modificar el código como sigue:

```
<select id="contactos">
        <optgroup label="Correos Públicos">
            <option value="info@danysoft.com"
    label=" Información en Danysoft" />
            <option value="admin@danysoft.com"
    label=" Administración de Danysoft" />
        </optgroup>
        <optgroup label="Correos Personales">
            <option value="Editor@danysoft.com"
    label=" Responsable de Publicaciones" />
            <option value="Ventas@danysoft.com"
    label=" Responsable de Ventas" />
        </optgroup>
</select>
```

Y con esta opción se generará una salida como la de la figura 12, una vez seleccionado el elemento:

Fig. 12: Opción equivalente a la anterior, pero agrupada mediante <optgroup>, vista en IE10

En tiempo de ejecución, si lo ejecutamos en IE10, no solo tendremos esta funcionalidad, sino que, además, el navegador efectúa dos animaciones en los procesos de despliegue y cierre, respectivamente, dando así un toque de fluidez a la interfaz de usuario.

Esto es un comportamiento predeterminado del motor de este navegador y no requiere ningún tipo de configuración por parte del usuario ni el uso de JavaScript.

Validaciones

Hoy día, tanto si hablamos de páginas que recogen información de los usuarios, como si se trata de aplicaciones Web, los mecanismos de validación son básicos si queremos evitar viajes innecesarios al servidor durante el proceso. El estándar aporta unas cuantas -que ya hemos visto-, pero también sugiere otras, a veces acompañadas de alguna API de JavaScript para permitir un ajuste más fino de los datos a validar antes de enviarlos al servidor.

El atributo *required*

Uno de los más básicos es el de exigir la presencia de información en un elemento de entrada. Para ello, disponemos (para todos los valores posibles de ***type***), del atributo ***required.***

En teoría, el funcionamiento presupone que en los navegadores en los que esté implementado, no permitirá el envío de la información de un formulario al servidor si el campo requerido no contiene algo, y –como siempre- será cosa del agente de usuario escoger una interfaz adecuada que avise de esta situación.

Dentro del navegador IE10, la interfaz es similar a los casos anteriores de avisos de error, presentando una etiqueta flotante (configurable) que impide el envío e indica el problema, como podemos ver en la figura 13.

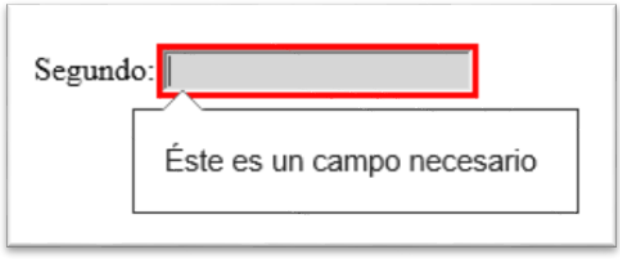


Fig. 13: Información de validación en IE10

La forma de personalizar el mensaje ofrecido, la veremos a continuación, al hablar de las funciones para validación.

Otra opción de validación es la multiplicidad. El atributo ***multiple***, indica al navegador que para ciertos valores del atributo ***type***, se permite una lista de entradas separadas por comas.

Los atributos novalidate y formnovalidate

En los casos en que la validación pudiese ser un problema, podemos utilizar el atributo ***novalidate***, disponible en elementos ***form*** con restricciones de validación para permitir envíos sin validaciones.

Podría ser que tuviésemos varios botones de envío en un formulario; por ejemplo, uno para enviar el formulario y otro para guardar una versión parcialmente completada del mismo a fin de que el usuario pueda terminar de rellenarlo más tarde. También se puede hacer que el botón de envío principal valide elementos de formulario con restricciones de validación y evitar que el botón "guardar para después" valide dichos elementos.

Para ello, en vez de establecer el atributo ***novalidate*** en todo el formulario, asignamos el atributo ***formnovalidate*** al botón que guarda el formulario parcialmente completado. Esto permite gran flexibilidad, y puede combinarse con las API de almacenamiento local (***localStorage***) de forma sencilla, para evitar que se pierdan los datos introducidos.

El siguiente ejemplo, modificado a partir de la documentación publicada por MSDN, se muestran ambos atributos en uso. El atributo ***formnovalidate*** se establece para el botón **Guardar para después**, mientras que el atributo ***novalidate*** se establece en el segundo elemento ***form***. Esto significa que, en el primer formulario, el botón **Enviar** validará que esos elementos del formulario tengan restricciones de validación, pero que el botón **Guardar para después** no. En el segundo formulario, no se validará ningún elemento, independientemente de las restricciones establecidas.

```
  <form>
    <p>Área con validaciones</p>
<p><label>* Nombre: <input type="text" required
/></label></p>
```

```
<p><label>* Calle: <input type="text" required
/></label></p>
    <p><label>* C.Postal: <input type="text"
pattern="[0-9](5)" placeholder="5 dígitos numéricos"
required="required" title="Campo de 5 dígitos"/>
</label></p>
    <input type="submit" name="btnEnviar" value="Enviar"
/>
<input type="submit" formnovalidate name="btnGrabar"
value="Guardar para más tarde" />
    <p>* Campo obligatorio</p>
  </form>
  <form novalidate>
    <p>Área no validada</p>
<p><label>Nombre: <input type="text" required
/></label></p>
<p><label>Calle: <input type="text" required
/></label></p>
<p><label>C.Postal: <input type="text" pattern="[0-9](5)"
placeholder="5 dígitos numéricos"
required="required" title="Campo de 5 dígitos"
/></label></p>
    <input type="submit" name="btnGrabarNV"
value="Enviar" />
  </form>
```

El comportamiento en ejecución es similar a ejemplos anteriores, presentando los mismos mensajes en caso de que no se cumplan las condiciones. Precisamente por eso, un paso necesario consiste en comprobar cómo podemos personalizar esos mensajes con la ayuda de pseudo-clases de CSS, como vimos en el ejemplo de los marcadores de posición.

Por ejemplo, si estamos utilizando un elemento ***<input>*** tipo ***password***, y, durante la introducción de los datos tenemos activada la tecla de mayúsculas, IE10 mostrará el siguiente mensaje error:

Fig. 14: Mensaje de error predeterminado de IE10

A continuación vamos a ver algunas posibilidades de configuración de esos mensajes de error.

Personalización de los mensajes de error y las API de validación

Para personalizar los mensajes de error basta con incluir un valor propio en el atributo ***title*** del elemento y éste contenido se anexará al texto predeterminado complementando la información. Es importante, que si queremos ese funcionamiento, incluyamos el patrón (***pattern***) que servirá de mecanismo de evaluación.

Además de ésta opción, y como podemos ver en la página oficial de Microsoft para el ejemplo anterior[40], es posible llegar más allá en la personalización haciendo que la interfaz visual de las validaciones utilice dos pseudo-clases CSS: las reglas ***:valid*** e ***:invalid***. Estas reglas se completan con otro par de pseudo-clases que permiten el control para el atributo ***required***: las reglas ***:optional*** y ***:required***.

Si definimos aspectos visuales distintos para estos dos pares de casos, mientras el valor no sea el correcto, se mostrará la interfaz visual establecida para ***:invalid*** y ***:required***, y esto se comprobará cada vez que el usuario introduce un carácter en la caja de texto.

La misma página, sugiere un ejemplo para la comprobación de estas opciones. Nos basamos en él, para adaptarlo, extenderlo y proponer el

[40] http://msdn.microsoft.com/es-es/library/hh673544(v=vs.85)

siguiente código de comprobación de estas 4 pseudo-clases:

```
  <style type="text/css">
  #frmValidaciones input:valid {
    border :solid lime;
    font-weight:normal;
  }
  #frmValidaciones input:invalid {
    border:solid red;
    font-weight:bold;
  }
  #frmValidaciones input:required{
    background-color:Yellow;
  }
  #frmValidaciones input:optional{
    background-color:LightGray;
  }
  </style>
</head>
<body>
  <form id="frmValidaciones">
    <p><label>Introduzca texto: <input
="text"/></label></p>
    <p><label>*Dirección de correo válida:
        <input type="text" required="required"
               value="correo@dominio.ext"/>
       </label></p>
    <p><label>URL válida:
        <input type="url" value="NO es una URL.
Introduzca el protocolo http"
pattern = "^http\://[a-zA-Z0-9\-\.]+\.[a-zA-
Z]{2,3}(/\S*)?$" title="http://dominio.ext" size="50" />
       </label></p>
    <p>* campo requerido</p>
    <br /><br /><br /><br />
    <input type="submit" value="Enviar" />
  </form>
</body>
```

Más adelante, mientras la URL introducida no pase el patrón de

validación, el borde estará marcado en rojo, el fondo en gris, y aparecerá el mensaje correspondiente al intentar enviarlo, como aparece en la figura adjunta:

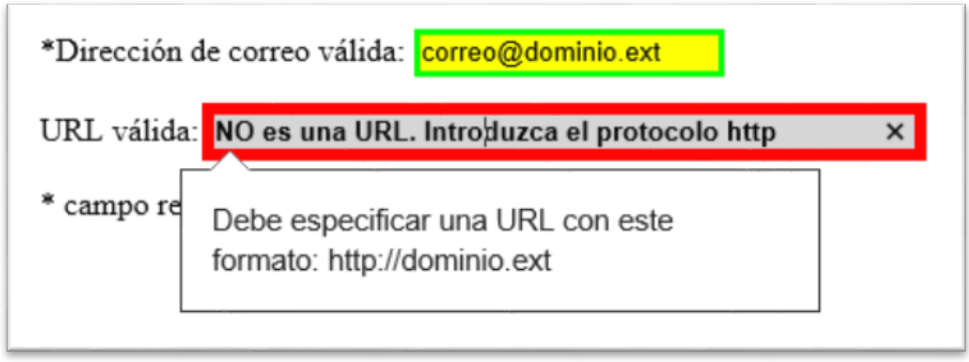


Fig. 15: Salida del código anterior al intentar *enviar el formulario*

Además de estas opciones, es posible establecer el mensaje completo personalizado que queremos que muestre la interfaz de usuario, mediante el API de Validación. Los estilos definidos al principio se aplican a cualquier elemento ***<input>*** contenido en el formulario ***frmValidaciones*** y hemos usado una expresión regular de para URL en la comprobación. El valor del atributo ***title*** se añade al mensaje predeterminado en caso de fallar la validación.

La API de Validación

HTML5 presenta un nuevo conjunto de mecanismos para validación basados en API. El objeto fundamental para esa estructura es ***validityState***. Puede ser accedido desde cualquier formulario en un navegador que soporte este tipo de validación, por supuesto. Cada elemento de un formulario (o el propio formulario) puede llamar al método **checkValidity()** para obtener un objeto de esta clase y analizar su contenido con la idea de mostrar al usuario la información pertinente sobre validaciones.

O también podemos llamar al método para comprobar la validez en general. Por ejemplo, si añadimos una función JavaScript a la página anterior, podemos llamar a esa función de la siguiente forma:

```
<script>
function Comprobar() {
   var formulario =
document.getElementById("frmValidaciones");
   var resultado =
(formulario.checkValidity()) ? "Válido" : "No Válido";
   alert(resultado);
 }
 </script>
…
<input type="submit" value="Enviar"
onclick="Comprobar();" />
```

Con esto, disponemos de otro mecanismo alternativo de validaciones, más personalizable, que generaría la salida esperada por pantalla:

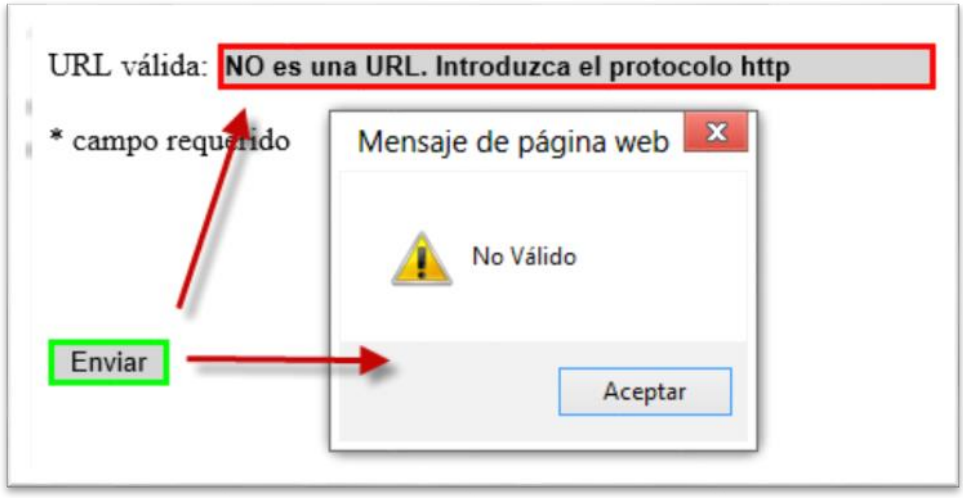


Fig. 16: Caja de diálogo resultado de la llamada a la función Comprobar en el código anterior

Este objeto se expone a través del atributo ***validity*** que cada formulario contiene. Puede adoptar una lista de valores dependiendo del tipo de error de validación que se haya producido (si se ha producido alguno). Los valores son los siguientes:

- **valueMissing** (campo requerido)

- **typeMismatch** (error de tipo)
- **patternMismatch** (patrón incorrecto)
- **tooLong** (demasiado largo)
- **rangeUnderflow** (rango por debajo de lo esperado)
- **rangeOverflow** (rango excedido)
- **stepMismatch** (paso intermedio incorrecto)
- **customError** (error personalizado)
- **valid** (no se cumple ninguna de las anteriores)

Todos estos valores podrán ser consultados más adelante utilizando funciones JavaScript, de forma que podamos ajustar exactamente el mensaje al tipo de error que se ha producido.

Para comprobar este extremo, veamos a continuación un ejemplo global que pone a prueba estas propiedades, ofreciendo una introducción de un valor numérico dentro de un rango, y comprobando acto seguido cuáles son los valores que adopta el objeto ***validity*** para cada situación:

```
<script type="text/javascript">
   function obtenerEstado() {
     var objetoES =
     document.getElementById("campoEntrada");
     var estadoObjetoES = objetoES.validity;
     var objetoIU =
     document.getElementById("mostrarEstado");
     objetoIU.innerHTML = "";
     // Recorrer los atributos
     for (var atributo in estadoObjetoES) {
         objetoIU.innerHTML += (atributo + ": <strong>" +
         estadoObjetoES[atributo] + "<strong><br/>");
         }
     }
</script>
<body onload="obtenerEstado();">
<h1>Prueba del objeto validityState</h1>
<div>El campo de texto solo acepta números entre 3 y 30
en incrementos pares. <br /> Pruebe números distintos
para comprobar los valores que adoptan los <br />
```

```
atributos del objeto
<strong><em>validityState</em></strong>
</div>
<br />
<div><label>Introduzca un número entre 3 y 30:
<input id="campoEntrada" type="number" required  min="3"
max ="30" step="2" />
<button onclick="obtenerEstado();">Comprobar
Estado</button>
</label>
</div>
<div id="mostrarEstado"></div>
</body>
```

En el bucle ***for***, recorremos el conjunto de atributos del objeto ***validity*** obtenido previamente. Cada llamada provoca una nueva validación, que dependerá del valor contenido en el campo de texto en ese momento. En la figura 17 podemos ver la salida de esta página según se evalúa el número 31:

Fig. 17: Salida del código anterior en IE10 tras comprobar un valor concreto.

En cuanto a la forma de personalizar el mensaje de error, podemos

utilizar la propiedad ***setCustomValidity***, mediante código JavaScript. Por ejemplo, podríamos escribir el siguiente código junto a un *script* de validación:

```
<form id="frmMensajesPersonalizados">

  <label>
      Introducza una palabra (excepto "byte"):
<input id="sinByte" type="text"
oninput="comprobar(this)" required="required" />
     <input type="submit" value="Enviar" />
   </label>
<div>
<button onclick="verErrores();">Previsualizar
errores</button>
</div>
<div id="mensajeError"></div>
</form>

// Script de validación
<script>
    function comprobar(input) {
        if (input.value == "byte") {
input.setCustomValidity("El texto -" + input.value + "-,
no es válido");
        } else {
            // Si es correcto anulamos la asignación
            input.setCustomValidity('');
        }
    }
    function verErrores() {
        var frm = document.getElementById("sinByte")
        document.getElementById("mensajeError").innerHTML
                                    = frm.validationMessage;
    }
</script>
```

De esa forma, si se produce cualquier situación en la que es importante modificar el mensaje predeterminado, llamamos a la función

setCustomValidity, pasándole el texto que nos interese, sabiendo que, cada vez que se introduce texto, se comprueba la situación en el navegador local. Si existe un error, se genera la salida que vemos en la figura 18:

Fig. 18: Mensaje de error personalizado en IE10

Obsérvese que el atributo **validationMessage** devuelve el mensaje de error que se muestra en función del estado actual del formulario. Además, todos los elementos disponen (desde las API de JavaScript), de un atributo ***willValidate***, que nos indica si el elemento pasará por un proceso de validación tal y como está configurado en ese momento.

Soporte en Visual Studio 2012/2013

Como vamos indicando para todos los elementos, el soporte de Visual Studio 2012 es completísimo, y podemos usar todas las opciones habituales con otros lenguajes, extendiéndolas a JavaScript, ya que el editor y los depuradores reconocen todas las nuevas API y atributos del estándar.

Por ejemplo, cuando vamos a comprobar esta opción desde el editor de código JavaScript, observamos cómo éste reconoce todos estos nuevos elementos de las API de validación (Ver figura 19):

Fig. 19: Soporte de las API de validación en Visual Studio 2012/2013

El nivel de implementación de estas API es muy completo en las últimas versiones de todos los navegadores que analizamos aquí.

El conjunto de atributos form*

Como parte de la idea motriz de permitir la personalización de los elementos y su comportamiento, han aparecido varias familias de atributos, que comparten sintaxis y nomenclatura. Este es el caso de la personalización de formularios. Para ello, los elementos de entrada **input**, **output**, **select**, **textarea**, **button**, **label**, **object** y **fieldset**, disponen ahora de un atributo ***form*** que permite indicar –asignando el ***id*** del formulario- uno o más formularios a los que pertenece el elemento (en caso de querer especificar más de uno, se utilizan una lista separada por comas), sin necesidad de que el elemento sea descendientes del formulario y pudiendo estar en cualquier parte de la página.

Esto tiene repercusiones en la estructura del documento, ya que permite reubicar los elementos de manera más adecuada, permitiendo que la lógica de procesamiento sea la misma.

El ejemplo oficial tiene este aspecto:

```
<label>
```

```
    Phone: <input type="tel" form="myForm" name="myTel"
/>
</label>
<form action="demo.aspx" method="get" id="myForm">
```

Donde la etiqueta ***label*** no forma parte de los descendientes del formulario ***myForm*** y, sin embargo, a efectos de proceso, se establece un vínculo entre ambos.

Relacionado con lo anterior, la familia de atributos con prefijo ***form**** está pensada para sobrescribir los valores aplicados a su ***<form>*** contenedor (solo es válido para los elementos ***input de tipo botón***, como ***button*** o ***submit***, o para el elemento ***button***). La lista completa y su funcionalidad básica es la siguiente:

Atributo	Descripción	Efecto
formaction	Sobrescribe el atributo ***action***	
formenctype	Sobrescribe el atributo ***enctype***	
formmethod	Sobrescribe el atributo ***method***	Forma de envío. Puede valer ***get*** o ***post.***
formnovalidate	Sobrescribe el atributo ***novalidate***	Anula la validación en el envío de un formulario
formtarget	Sobrescribe el atributo ***target***	

Tabla 1: Conjunto de atributos nuevos para formularios

En el código siguiente, aunque el mismo formulario asigna el atributo ***action*** a "*demo.aspx*", el elemento ***OtroEnvio*** llamará a la página "*demo_adm.aspx*", pensada para administradores. Además, el último elemento ***input*** no generará ningún tipo de validación (aunque sí enviará los datos al servidor).

```
<form action="demo.aspx" method="get" id="myForm">
    Correo:
```

```
    <input type="email" name=" correoUsuario" />
    <input type="submit" value="Enviar" />
<input type="submit" id="OtroEnvio"
formaction="demo_adm.aspx" value="Enviar a
administradores" />
    <input type="submit" formnovalidate="formnovalidate"
           value="Envío sin validación" />
</form>
```

Todo esto facilita una disposición lógica de los elementos independiente del formulario al que pertenezcan y también favorece el tratamiento de situaciones excepcionales.

Nuevos Atributos de carácter global

Hasta aquí hemos hablado de nuevos atributos, tanto para los elementos nuevos como para los antiguos. Pero la mayor parte de las veces, se trata de atributos específicos de elementos específicos. No obstante, existen ahora varios conjuntos de atributos de carácter global (pueden formar parte de cualquier elemento), como es el caso de los relacionados con la accesibilidad (**WAI**-**ARIA**), los atributos personalizados **data-***, etc...

Atributos que ahora tienen carácter global

En primer lugar, varios atributos de HTML4 ahora se aplican a todos los elementos: son los atributos globales: ***accesskey***, ***class***, ***dir***, ***id***, ***lang***, ***style***, ***tabindex*** y ***title***. Además, XHTML 1.0 sólo permite ***xml:space*** en algunos elementos, pero ahora está permitido en todos los elementos de los documentos XHTML.

Pero también hay varios atributos globales nuevos:

- El atributo ***contenteditable*** que indica que el elemento es un área editable. El usuario puede cambiar el contenido del elemento y manipular el marcado.

- El atributo ***contextmenu***, que puede utilizarse para señalar un menú contextual programado por un desarrollador Web (de momento, no soportado por ningún navegador).
- La colección de atributos ***data-****, definidos por programación. Los desarrolladores web pueden definir cualquier atributo que quieran, siempre que comience con el prefijo **data-,** para evitar colisiones con futuras versiones de HTML. Se utilizan para almacenar datos personalizados que puedan ser consumidos por la página web o la propia aplicación. No están destinados a ser usados por otras partes (por ejemplo agentes de usuario).
- Los atributos ***draggable*** y ***dropzone*** que pueden utilizarse junto con la nueva API ***Drag&Drop*** (Los estudiaremos en el Capítulo 6).
- El atributo ***hidden*** indica que un elemento no está todavía disponible, o ya no es relevante.
- Los atributos ***role*** y la colección ***aria-**** pueden utilizarse para configurar tecnología accesible. Estos atributos tienen requerimientos ligeramente distintos en WHATWG HTML y W3C HTML5/W3C HTML5.1.
- El atributo ***spellcheck*** establece si puede comprobarse contenido de ortografía o no.
- El atributo ***translate*** da una pista a los traductores sobre si el contenido debe ser traducido o no.

Los atributos *WAI-ARIA*

Se trata de un conjunto de atributos especialmente pensado para la gestión de la accesibilidad en agentes de usuario. A partir de ahora, todos los elementos disponen de un conjunto completo de atributos precedidos por el prefijo ***aria-***, indicando cada uno de ellos un aspecto de accesibilidad (ARIA significa "*Accesible Rich Internet Applications*").

La especificación WAI-ARIA 1.0 está pensada para conseguir nuevas formas de comunicar contenidos, y alcanzó en enero de 2011 el estatus

de "*Candidate Recommendation*"[41], si bien, para una explicación más detallada de las mejores prácticas para desarrolladores, recomendamos el documento "*WAI-ARIA 1.0 Authoring Practices. An author's guide to understanding and implementing Accessible Rich Internet Applications*"[42], donde se explican los detalles de implementación con ejemplos de uso de múltiples casos.

Esta especificación sugiere un total de 35 atributos específicos de accesibilidad, cubriendo todas las opciones necesarias de los programas lectores de pantallas Web.

A estos atributos, hay que sumar el atributo ***role***[43]**,** que, en este contexto tiene un papel fundamental. Hace las veces de puente entre los lectores de pantalla (y otros agentes de accesibilidad) y las interfaces de usuario, permitiendo indicar el tipo de control de usuario que se representa, para "conectar" la interfaz habitual, con la ofrecida por los agentes mediante algún mecanismo adecuado.

Por ejemplo, un selector de color, (elemento del tipo **<input type="color">**), podría no ser entendido por un lector de pantalla. De esta forma, se añade un componente semántico que indica cuál es la verdadera función del elemento en el contexto de la página o aplicación[44].

Dicho de otra forma, le aporta pistas al mecanismo que utilice el intérprete de la página para que pueda hacer corresponder los distintos elementos HTML con los suyos propios, y facilite la "traducción" de la interfaz visual de usuario a otra manejable por esos lectores.

Visual Studio (desde 2010), no muestra la lista completa de atributos

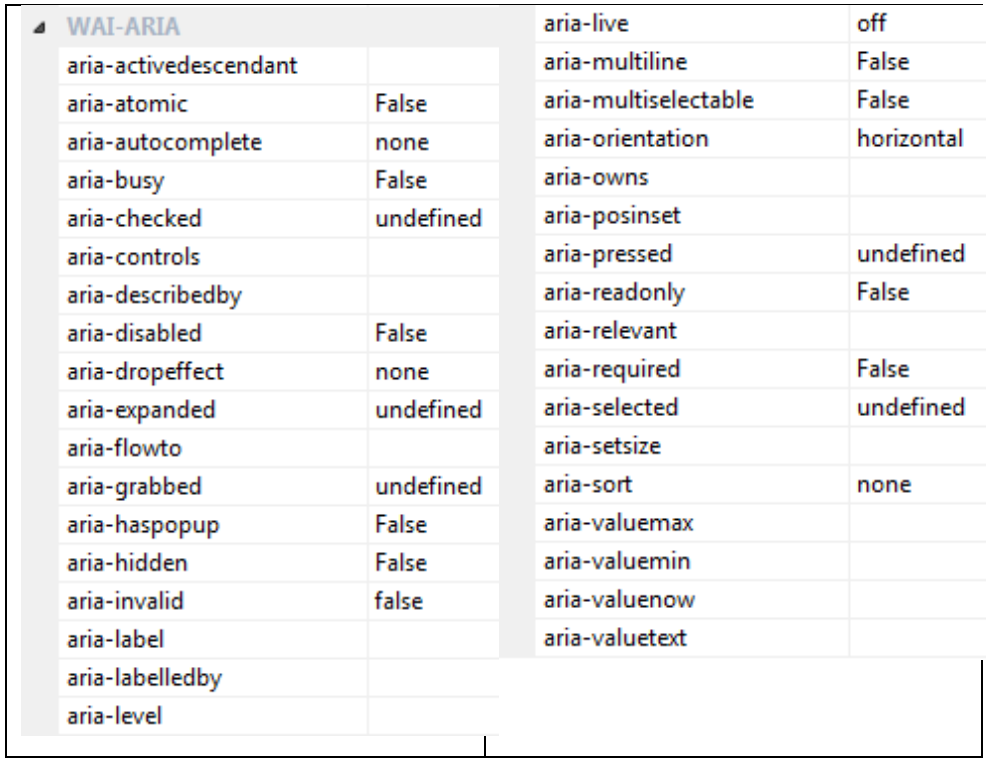
41 http://www.w3.org/TR/wai-aria/

42 http://www.w3.org/TR/wai-aria-practices/

43 Ya presente en la especificación XHTML 1.1: http://www.w3.org/TR/xhtml-role/#s_role_module_attributes

44 En el sitio Web "Developer.mozilla.org" se dispone de una tabla completa de correspondencia de los distintos roles disponibles y como debieran ser interpretados: https://developer.mozilla.org/en/ARIA_to_API_mapping

disponibles en la Ventana de Propiedades de cualquier elemento HTML, tal y como vemos en la figura 20.



WAI-ARIA	
aria-activedescendant	
aria-atomic	False
aria-autocomplete	none
aria-busy	False
aria-checked	undefined
aria-controls	
aria-describedby	
aria-disabled	False
aria-dropeffect	none
aria-expanded	undefined
aria-flowto	
aria-grabbed	undefined
aria-haspopup	False
aria-hidden	False
aria-invalid	false
aria-label	
aria-labelledby	
aria-level	
aria-live	off
aria-multiline	False
aria-multiselectable	False
aria-orientation	horizontal
aria-owns	
aria-posinset	
aria-pressed	undefined
aria-readonly	False
aria-relevant	
aria-required	False
aria-selected	undefined
aria-setsize	
aria-sort	none
aria-valuemax	
aria-valuemin	
aria-valuenow	
aria-valuetext	

Fig. 20: Lista completa de atributos WAI-ARIA tal y como se ven en Visual Studio

El código siguiente es un pequeño ejemplo basado en el oficial, de cómo se utilizarían estos atributos. Vemos la presencia del atributo ***aria-labelledby*** apuntando al control responsable de la funcionalidad estándar, junto al atributo ***role*** que le indica al lector el papel que representa ese elemento en el código de marcado.

```
<span role="checkbox" id="cb">Prefiere la marca de coche
<input type="text" aria-labelledby="cb" /></span><br
/><br />
<span role="checkbox" id="Span1"><input type="text" aria-
labelledby="cb" /> es la marca elegida</span><br /><br />
```

```
<span role="checkbox" id="Span2">Conduce un <input
type="text" aria-labelledby="cb" /></span>
```

También se le indica al agente de usuario los detalles de implementación en forma de metadatos, permitiendo que cada uno seleccione el más conveniente a presentar en cada situación.

Esos roles, pueden adoptar diversos valores en función del objetivo que se persiga, incluyendo aspectos de tipo semántico, en los que se denominan *Document Landmark Roles* (Roles de zonas del documento). Una lista típica de tales roles es la siguiente:

- **rol** = "*banner*" debe ser asignado a los elementos que contengan contenido orientado en el sitio, como el nombre del sitio, título y/o logotipo.
- **rol** = "*navigation*" debe asignarse al elemento que contiene vínculos de navegación.
- **rol** = "*main*" debe asignarse al contenido principal.
- **rol** = "*search*" debe asignarse al elemento que contiene el buscador.
- **rol** = "*article*" se asigna a contenido que es independiente; tiene sentido fuera del contexto del resto del documento.
- **rol** = "*complementary*" se asignan a contenido que es complementario al contenido principal.
- **rol** = "*contentinfo*" debe asignarse al contenido secundario de metadatos, como las notas y los derechos de autor.
- **rol** = "*application*" se asigna a las regiones de la página que incluyen el contenido de la aplicación y su funcionalidad. Tenga en cuenta que este papel es único por dos razones:
    - 1) si se asigna a la página completa, no se trata como un hito de navegación y
    - 2) Le indica a los lectores de pantalla que cambien entre el "modo de navegación" y el "modo de interfaz de aplicación".

De esta forma, esto no solo sirve para aplicaciones Web, sino en sitios –incluso existentes- para mejorar de forma sencilla la accesibilidad. Por ejemplo, si tenemos una lista ordenada que se transforma en un menú de navegación, bastaría con añadir uno de estos roles (el segundo de la

lista) para aportar un dato de accesibilidad:

```
<ul role="navigation">
    <li><a href="/">Inicio</a></li>
    <li><a href="/AcercaDe/">Acerca de...</a></li>
    ...
</ul>
```

Esto es de gran utilidad para permitir que estas herramientas habiliten el acceso a los típicos *widgets* de comunicación social. Por ejemplo, si tenemos un elemento de entrada para un correo electrónico, podemos indicar que ese elemento es obligatorio mediante el sencillo código fuente:

```
<input type="text" name="email" aria-required="true" />
```

Otro punto importante es el de las llamadas "Regiones activas", definidas por el atributo ***aria-live***. Una región activa es un fragmento de código de marcado del que se espera que reciba actualizaciones frecuentes. Consideremos el siguiente código fuente:

```
<p aria-live="polite" >...</p>
<p aria-live="assertive">...</p>
<p aria-live="off">...</p>
```

En el primer párrafo, *polite* indica una actualización cuando el usuario ha completado una actividad. En el segundo, se indican notificaciones de una actualización de forma inmediata (aunque debiera utilizarse con cuidado, ya que interrumpe la tarea actual del usuario), y el en tercero, se desactivan todas las notificaciones.

Todas estas aportaciones pueden complementarse con código JavaScript, para conseguir el objetivo de un sitio realmente accesible. Hemos incluido varias referencias a información adicional en el apartado final de este capítulo.

El atributo contenteditable

Se trata de la posibilidad de permitir al lector la edición de un elemento de la página. Según la especificación, el usuario podría cambiar tanto el contenido como la etiqueta.

Es un dato de tipo enumerado, cuyos valores pueden ser la cadena vacía, *true* y *false*. Los dos primeros se corresponden con el estado ***true***. ***False*** genera un estado ***false***, lógicamente, y existe un tercero, ***inherit*** que es el valor asociado cuando no existe el atributo o su valor no es correcto.

Si ***true*** indica que el contenido es editable, ***inherit*** presupone que el contenido puede editarse, si su contenedor lo es. Funciona en conjunción con el antiguo atributo ***designMode*** que establece condiciones de funcionamiento (de edición, realmente) a nivel de página.

Como podemos ver en la figura 21 el soporte en Visual Studio (edición/ejecución) es completo:

Fig. 21: Soporte de contenteditable en Visual Studio 2012/2013

Una vez establecido en ***false***, el comportamiento de IE10 impide que el elemento tome el foco, y por tanto, que sea editable.

El atributo *hidden*

Se trata de un atributo ya implementado por los navegadores en versiones anteriores, pero que no había alcanzado el carácter de "oficial" en el estándar. No debe confundirse con el elemento ***<input type="hidden">*** que se usa en HTML 4.

Su propósito, es el de ocultar un elemento indicando que ya no resulta relevante, y no debiera ser mostrado por el agente de usuario. Su sintaxis,

es muy simple:

```
<span hidden="hidden">Datos</span>
```

El atributo *spellcheck*

Indica si, para un elemento dado, debiera de comprobarse su sintaxis utilizando un diccionario disponible. Se trata un dato *booleano*, así que sus únicos valores asignables son ***true*** y ***false***.

Afecta a 3 elementos sobre los que es posible ejecutar un análisis: los elementos **<*input*>** de tipo ***text***, el texto en elementos **<*textarea*>**, y el texto de cualquier elemento marcado como editable con cualquiera de los atributos comentados en el punto anterior. La sintaxis sería algo como lo siguiente:

```
<label for="Segundo">Segundo: <input id="Segundo"
spellcheck="true" />
</label>
```

Y la salida visual en IE10/11 es bastante evidente, ya que el analizador va detectando las pulsaciones, y cuando aparece un espacio en blanco, analiza la palabra anterior, subrayándola en rojo (esto es configurable), tal y como vemos en la figura 22:

Fig. 22: IE10 marcando un error en texto de entrada cuyo atributo spellcheck está activado

El soporte de los navegadores es bastante bueno y, salvo FireFox, todos los analizados (incluyendo IE9) la soportan sin problemas.

Los atributos draggable y dropzone

El primero de ellos indica si un elemento puede arrastrarse mediante una intervención del usuario, y el segundo declara -para un elemento dado- la acción a realizar en caso de que tenga lugar una acción de soltar dentro de su zona: si se debe copiar, mover o enlazar el contenido arrastrado.

Su funcionamiento se encuentra directamente relacionado con las nuevas API de JavaScript, de forma que lo estudiaremos en el capítulo 6º.

Los atributos *data-**

Otra de las novedades que apuntábamos al principio es la posibilidad de crear atributos personalizados que pueden posteriormente ser manejados mediante JavaScript.

Esta característica la hemos dejado para el final porque es una de las más importantes de entre las novedades de HTML 5 de cara a los desarrolladores de aplicaciones Metro Style para Windows 8, como veremos a continuación.

La especificación indica que se trata de un atributo que no pertenece a ningún espacio de nombres, comienza con el prefijo ***data-*** y tiene al menos un carácter después del guion. Si se cumplen esos requisitos, el atributo será tratado como un área de almacenamiento para datos propios. Obviamente, no afectan a la presentación en sí, ya que no tienen ninguna correspondencia visual.

La idea se parece mucho a lo que en otros lenguajes orientados a objetos se denominan "propiedades de usuario" (por ejemplo la propiedad ***Tag*** en controles de .NET Framework), y permite almacenar datos propios para los que no existen en la especificación una etiqueta o mecanismo adecuados.

En el código siguiente, añadimos 3 atributos personalizados a una etiqueta <***span***>, y posteriormente les recogemos y mostramos en pantalla a petición del usuario (al pulsar el botón).

```
<span id="SobreDanysoft"
     data-ubicación="Madrid"
     data-actividades="Formacion, Consultoría,
       Publicaciones, Certificaciones Certificaciones y
       Software"
     data-sitioWeb="www.danysoft.com">
    <b>Danysoft es un grupo español, que desde 1990 está
especializado en asesorar en la adquisición e inversión
en tecnologías de la información a clientes de España,
Portugal y Latinoamérica, así como ofrecer servicios de
formación, instalación/configuración y consultoría
necesarios para lograr su correcta implantación en los
diferentes entornos de sus clientes.</b>
</span><p />
<input id="btnLeer" type="button" title="Saber más acerca
de Danysoft" value="Más información..."
onclick="DanysoftInfo()" />
<p><span id="masInformacion">Más información</span></p>

<script type="text/javascript">
    var custElement =
document.getElementById("SobreDanysoft");
     var info =
document.getElementById("masInformacion");
     function DanysoftInfo() {
         // Recuperamos información mediante una llamada
         // a getAttribute;
         masInformacion.innerHTML = 'Ubicación: ' +
         custElement.getAttribute("data-ubicación") +
"<br/>";
         masInformacion.innerHTML += 'Actividades: ' +
         custElement.getAttribute("data-actividades") +
"<br/>";
         masInformacion.innerHTML +=
    "Sitio Web: <a target='_blank' href='http://" +
         custElement.getAttribute("data-sitioWeb") + "'>"
+
         custElement.getAttribute("data-sitioWeb") +
"</a><br/>";
     }
```

```
</script>
```

El código anterior funciona correctamente en cualquiera de los navegadores que utilizamos para las pruebas, con una salida (tras pulsar el botón), como la de la figura 23:

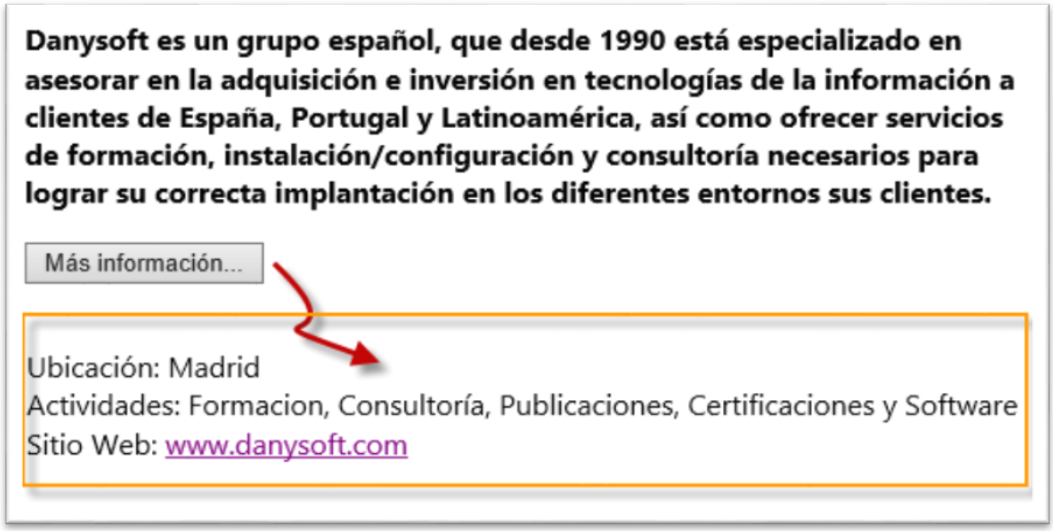


Fig. 23: Salida del código anterior en IE10/11, mostrando los valores de los atributos personalizados.

La importancia de este nuevo atributo, va mucho más allá de lo que podría suponerse a primera vista, y constituye buena parte del armazón que Microsoft ha construido en la creación de aplicaciones Web para aplicaciones Metro Style (y también para las otras), siempre que estén basadas en la tecnología del estándar HTML 5.

El atributo *data-* y las aplicaciones HTML5 para Windows 8

La razón es la enorme flexibilidad que ofrece esta opción, especialmente cuando va unida a una implementación del motor de JavaScript de última generación, tal y como sucede con el motor *Chakra* integrado en los exploradores IE9/10/11.

Mediante la conjunción de estos dos factores, las aplicaciones Windows 8 escritas en el estándar HTML5, pueden aprovechar una capa de interpretación muy optimizada, llamada ***WindowsRT*** (*Windows RunTime*), y comunicarse directamente con las API básicas del sistema operativo, de forma que el mismo código pueda suministrar experiencias casi idénticas en la Web (bajo un navegador), o en el escritorio, como sucede con las aplicaciones estilo Metro.

Un pequeño ejemplo puede servir perfectamente para ilustrar esto. Si seleccionamos una de las aplicaciones tipo Metro utilizando las plantillas predeterminadas de Visual Studio 2012, nos encontraremos con varios modelos básicos que sirven de punto de partida. Uno de ellos, es el modelo "Aplicación Dividida" en el que se disponen los elementos de la interfaz de usuario dentro de una rejilla controlada mediante estilos y JavaScript, y donde cada una de las celdas albergará los contenidos que queramos darle más adelante.

La aplicación consta de un conjunto de carpetas y archivos predeterminados que aportan la funcionalidad inicial. Podemos ver la lista en la figura 24:

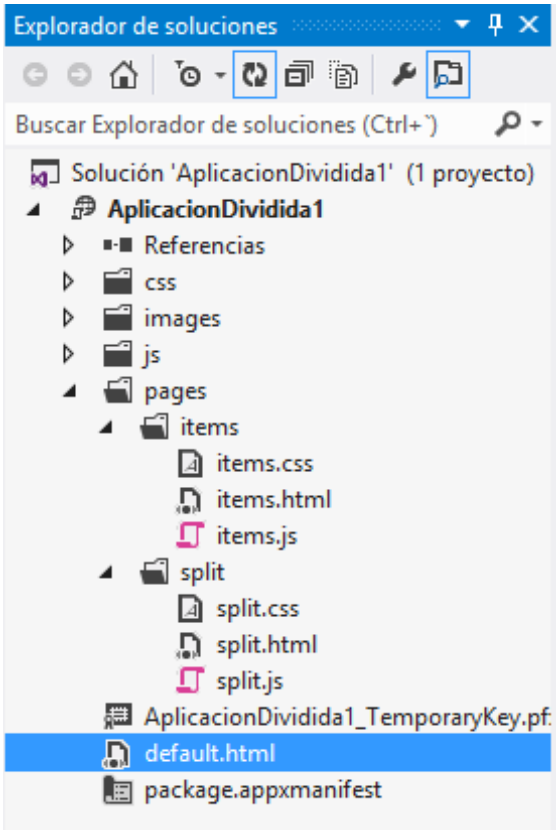


Fig. 24: El explorador de soluciones de V. Studio 2012/2013, mostrando la aplicación dividida inicial

Como puede verse, contiene una división de carpetas cuidadosamente organizada para almacenar los archivos necesarios para su funcionamiento (léase, las hojas de estilo y los archivos JavaScript que utilizará en ejecución). El código de marcado que se asocia con la pantalla principal es el incluido en "*default.html*".

Pues bien, éste es el código generado por visual Studio para el cuerpo de esa página:

```
<body>
<div data-win-control="Application.PageControlNavigator"
     data-win-options="{home: '/pages/items/items.html'}"
id="contenthost" >
```

```
</div>
     <!--
     <div id="appbar" data-win-control="WinJS.UI.AppBar">
<button data-win-control="WinJS.UI.AppBarCommand" data-
win-options="{id:'cmd', label:'Command',
icon:'placeholder'}"> </button>
    </div> -->
</body>
```

Como vemos, el atributo ***data-*** es utilizado por razones de almacenamiento y *databinding*, en este caso, para indicar, respectivamente, quién es el control manejador (***Application.PageControlNavigator***), y para establecer la ubicación de destino vinculada con el inicio (***home: '/pages/items/items.html***).

Esto significa que, posteriormente, el código JavaScript leerá la información asignada a esos atributos, utilizándola según interese. Muchos de esos nombres compuestos de atributos, disponen de un significado bien definido para las librerías de JavaScript del motor principal de la capa ***WindowsRT*** del sistema, que en este contexto se denomina ***WinJS.***

Esto facilita no poco la construcción de este tipo de aplicaciones y es algo que se repite igualmente en otras páginas del proyecto.

Existe ya abundante información en el sitio MSDN de Microsoft acerca de las aplicaciones Metro Style y los distintos modelos de construcción, por lo que remitimos al lector a esas fuentes para cualquier información adicional.

Los atributos vinculados a eventos

La especificación también indica que, ahora, todos los atributos que expresan vinculación con un manejador de evento (los que toman la forma ***onevento***), tienen ahora carácter global, incluyendo elementos que todavía no se encuentran en la especificación.

Esto incluye algunos eventos nuevos, como ***onplay***. La lista completa es

la siguiente:

• **onabort** • **onblur*** • **oncancel** • **oncanplay** • **oncanplaythrough** • **onchange** • **onclick** • **onclose** • **oncontextmenu** • **oncuechange** • **ondblclick** • **ondrag** • **ondragend** • **ondragenter** • **ondragexit** • **ondragleave** • **ondragover** • **ondragstart** • **ondrop** • **ondurationchange** • **onemptied** • **onended** • **onerror*** • **onfocus*** • **oninput** • **oninvalid** • **onkeydown** • **onkeypress** • **onkeyup** • **onload***	• **onloadeddata** • **onloadedmetadata** • **onloadstart** • **onmousedown** • **onmouseenter** • **onmouseleave** • **onmousemove** • **onmouseout** • **onmouseover** • **onmouseup** • **onmousewheel** • **onpause** • **onplay** • **onplaying** • **onprogress** • **onratechange** • **onreset** • **onscroll*** • **onseeked** • **onseeking** • **onselect** • **onshow** • **onsort** • **onstalled** • **onsubmit** • **onsuspend** • **ontimeupdate** • **onvolumechange** • **onwaiting**

Naturalmente, hay algunas consideraciones que tener en cuenta aquí. Por un lado, los atributos marcados con un asterisco (*) tienen un significado distinto si se especifican en un elemento ***body*** ya que estos exponen

manejadores de evento del objeto ***window*** con el mismo nombre.

Además, aunque estos eventos son aplicables a todos los elementos, no son útiles en todos ellos. Por ejemplo, un evento como ***volumechange*** solo funcionará en el caso de elementos multimedia.

Atributos modificados

También se han producido cambios (normalmente, extensiones o excepciones al funcionamiento) por coherencia con el resto de propuestas. Lo que sigue es una lista alfabética que describe los más importantes:

- El atributo de ***accept*** de ***input*** ahora permite los valores ***audio/****, ***video/**** e ***image/**** (además de los anteriores: un tipo ***mime*** sin parámetros o una cadena comience por un punto para indicar la extensión de un fichero).
- A los autores, se les sugiere que incluyan ambas opciones (tipo *mime* y extensión). Por ejemplo, una aplicación que convierta formatos de Microsoft Word en *Open Document Format*, podría incluir para la selección del fichero por el usuario el siguiente elemento ***input***:

```
<input type="file" accept=".doc,.docx,.xml,
application/msword,application/vnd.openxmlformats-
officedocument.wordprocessingml.document">
```

- El atributo ***accesskey*** global permite especificar múltiples conjuntos de caracteres que el agente de usuario puede elegir.
- El valor de atributo ***accesskey*** es utilizado por el agente de usuario como una guía para crear un atajo de teclado que se activa o se centra el elemento. En el ejemplo *oficial* siguiente, se dan una gran variedad de enlaces con las teclas de acceso para que los usuarios de teclado familiarizados con el sitio pueden navegar más rápidamente a las páginas pertinentes:

```
<nav>
    <p>
<a title="Consortium Activities" accesskey="A"
href="/Consortium/activities">Activities</a>
<a title="Technical Reports and Recommendations"
accesskey="T" href="/TR/">Technical Reports</a>
<a title="Alphabetical Site Index" accesskey="S"
href="/Consortium/siteindex">Site Index</a>
<a title="About This Site" accesskey="B"
href="/Consortium/">About Consortium</a>
<a title="Contact Consortium" accesskey="C"
href="/Consortium/contact">Contact</a>
    </p>
</nav>
```

- El atributo ***action*** en ***form*** ya no puede tener una URL vacía.
- En WHATWG HTML, el atributo ***method*** tiene una nuevo valor posible (***dialog***), para poder cerrar un elemento ***dialog***.
- El atributo ***border*** en ***table*** sólo permite los valores "1" y la cadena vacía. En WHATWG HTML, el atributo ***border*** está obsoleto.
- El atributo ***colspan*** en ***td*** y ***th*** ahora tiene que ser mayor que cero.
- El atributo ***coords*** en ***area*** ya no permite un valor porcentual del radio cuando el elemento se encuentra en el estado de círculo.
- El atributo ***data*** del elemento ***object*** ya está en relación con el atributo ***codebase***.
- El atributo ***defer*** en el elemento ***script*** ahora fuerza explícitamente a que el *script* se ejecute una vez terminado el análisis sintáctico de la página. Veremos más sobre este funcionamiento en el capítulo 5º dedicado a JavaScript.
- El atributo global ***dir*** permite ahora el valor ***auto***.
- El atributo ***enctype*** de un elemento ***form*** ahora soporta el valor ***text/plain***.
- A los atributos ***width*** y ***height*** en los elementos ***img***, ***iframe*** y ***object*** ya no se les permite contener porcentajes. Tampoco pueden usarse para expandir la imagen a una proporción de aspecto diferente a su relación de aspecto.
- El atributo ***href*** del ***link*** ya no puede tener una URL de vacía.

- El atributo ***href*** de ***base*** ahora puede contener una dirección URL relativa.
- Todos los atributos que admiten una URL, como, por ejemplo, ***href*** ahora admiten ***IRIs*** (***Internationalized Resource Identifiers***) si la codificación del documento es UTF-8 o UTF-16.
- Se entiende que el atributo ***http-equiv*** del elemento ***meta*** ya no debe ser utilizado por los servidores HTTP para crear encabezados HTTP en la respuesta. En su lugar, se sugiere una directiva ***pragma***.
- El atributo global ***id*** ahora puede tener cualquier valor, mientras sea único, no sea la cadena vacía y no contenga espacios.
- El atributo global ***lang*** admite la cadena vacía, además de un identificador de idioma válido, como sucede en ***xml:lang*** en ***XML***.
- El atributo de ***media*** en el elemento ***link*** ahora acepta un objeto multimedia y, por defecto, vale "***all***". El valor del atributo ***media*** debe estar conforme con uno de los valores indicados en la especificación "*Media Queries*"[45].
- Los atributos que definan un controlador de eventos (por ejemplo ***onclick***) ahora siempre deben usar JavaScript como el lenguaje de *scripting*.
- El atributo ***value*** del elemento ***li*** ya no es obsoleto ya que no se usa en presentación. Lo mismo ocurre con los atributos de ***start*** y ***type*** del elemento ***ol***.
- De hecho, este atributo merece un comentario, porque ahora es posible expresar dos salidas de listas idénticas, mediante dos sintaxis diferentes si queremos incluir numeración en la lista.
- Por ejemplo, si tenemos una simple lista de películas que queremos que aparezcan ordenadas según orden inverso podemos utilizar un elemento ***ol*** con el atributo ***reversed*** de la siguiente forma:

[45] http://w3c-test.org/csswg/mediaqueries3/

```
<figure>
    <figcaption>Las 5 mejores películas infantiles del
siglo</figcaption>
    <ol reversed>
        <li><cite>Cars</cite>, 2006</li>
        <li><cite>Toy Story 3</cite>, 2010</li>
        <li><cite>Up</cite>, 2009</li>
        <li><cite>Monsters, Inc</cite>, 2001</li>
        <li><cite>Ratatouille</cite>, 2007</li>
    </ol>
</figure>
```

Pero podemos obtener el mismo resultado pudiendo indicar los valores numéricos que deseamos en el atributo ***value***, lo que nos permite controlar la numeración:

```
<figure>
    <figcaption>Las 5 mejores películas infantiles del
siglo (v.2)</figcaption>
    <ol>
        <li value="5"><cite>Cars</cite>, 2006</li>
        <li value="3"><cite>Toy Story 3</cite>, 2010</li>
        <li value="3"><cite>Up</cite>, 2009</li>
        <li value="2"><cite>Monsters, Inc</cite>,
2001</li>
        <li value="1"><cite>Ratatouille</cite>, 2007</li>
    </ol>
</figure>
```

La salida es distinta, como podemos comprobar en la siguiente figura:

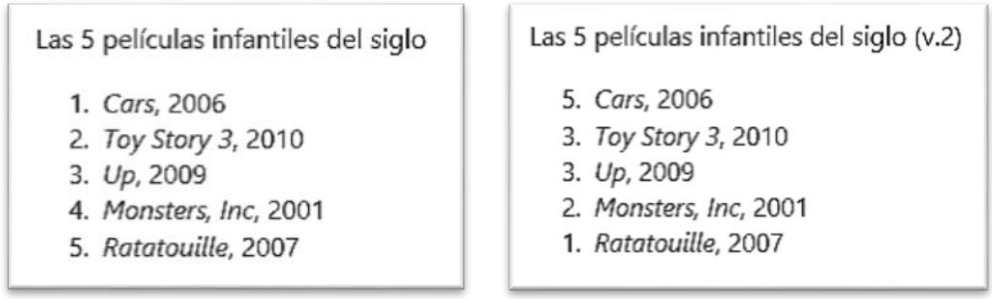


Fig. 25: En el segundo caso, tenemos control de la numeración

- El atributo global ***style*** siempre utiliza CSS como lenguaje de estilo.
- El atributo global ***tabindex*** ahora permite valores negativos para indicar que el elemento puede recibir el foco, pero no mediante la tecla *Tab*.
- El atributo de ***target*** de los elementos ***a*** y ***area*** ya no está obsoleto, al ser útil en aplicaciones Web, por ejemplo, en conjunción con ***iframe***.
- El atributo ***type*** en los elementos ***script*** y ***style*** ya no es necesario si el lenguaje es JavaScript y el lenguaje de estilo CSS, respectivamente.
- El atributo ***usemap*** en el elemento ***img*** ya no admite una dirección URL, sino que toma una referencia válida de formato ***#nombre*** a un elemento de ***map***.
- El documento oficial explica la situación con un ejemplo, que nosotros vamos a modificar y adaptar aquí, en el que se pone como premisa la existencia de 4 figuras geométricas que son parte del mismo gráfico (Ver figura adjunta):

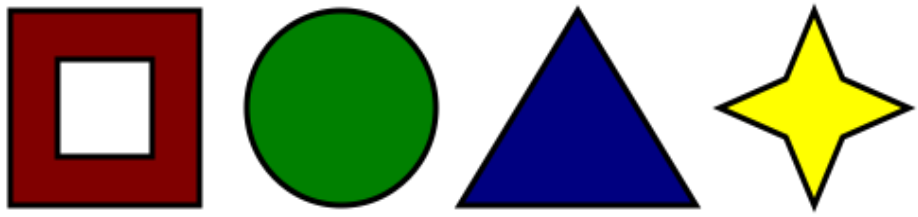


Fig. 26: Se desea que solo las partes coloreadas respondan a un evento click

- Se desea que el usuario pueda seleccionar una de las 4 figuras, pero solamente pulsando sobre las zonas coloreadas (y no sobre las blancas). La solución sería el código siguiente (el lector puede comprobarlo y verá que solo aparece el icono de selección en las zonas deseadas.

```
<p>
    Por favor, seleccione una figura:
    <img src="sample-usemap.png" usemap="#figuras"
         alt="Hay cuatro figuras disponibles: Un cuadrado
rojo, un círculo verde, un triángulo azul, y una estrella
amarilla de 4 puntas.">
    <map name="figuras">
        <!-- el agujero del cuadrado rojo -->
        <area shape="rect" coords="50,50,100,100">
        <area shape="rect" coords="25,25,125,125"
     href="rojo.html" alt="Cuadrado rojo.">
        <area shape="circle" coords="200,75,50"
     href="verde.html" alt="Círculo verde.">
        <area shape="poly" coords="325,25,262,125,388,125"
     href="azul.html" alt="Triángulo azul.">
        <area shape="poly"
coords="450,25,435,60,400,75,435,90,450,125,465,90,500,75
,465,60" href="amarillo.html" alt="Estrella amarilla.">
    </map>
   </p>
```

Observe el uso del atributo ***usemap*** de ***img*** enlazando con el

elemento ***map***, que define las áreas seleccionables.

Atributos no recomendados (y sus alternativas)

Además, se permiten los siguientes atributos, pero se disuade a los desarrolladores Web de usarlas y en cambio recomienda utilizar una solución alternativa:

- El atributo ***border*** en ***img***. Es necesario que tenga el valor "0" cuando esté presente. Los desarrolladores web deben utilizar CSS en su lugar.
- El atributo de ***language*** de ***script***. Es necesario que tenga el valor "JavaScript" (mayúsculas o minúsculas) cuando están presentes y no puede entrar en conflicto con el atributo ***type***. Los desarrolladores web simplemente pueden omitirla ya que no tiene ninguna función útil.
- El atributo ***name*** en ***a***. Los desarrolladores web deben utilizar el atributo ***id***, en su lugar.

Atributos obsoletos

Algunos atributos de HTML4 ya no están permitidos en HTML. La especificación define cómo los agentes de usuario deben procesarlos en documentos heredados, pero no se permiten a los desarrolladores Web utilizarlos y no se validarán correctamente.

HTML tiene consejos sobre lo que puede utilizar en su lugar.

- Los atributos ***rev*** y ***charset*** en ***link*** y ***a.***
- Los atributos ***shape*** y ***coords*** en ***a***.
- El atributo ***longdesc*** en **img** e **iframe**.
- El atributo ***target*** en ***link***.
- El atributo ***nohref*** en ***area***.
- El atributo ***profile*** en ***head***.
- El atributo ***version*** en ***html***.
- El atributo ***name*** en ***img*** (utilizar ***id*** en su lugar).

- El atributo ***scheme*** en ***meta***.
- Los siguientes atributos de ***object***: ***archive***, ***classid***, ***codebase***, ***codetype, declare*** y ***standby*** .
- Los atributos ***valuetype*** y ***type*** de ***param***.
- El atributo ***axis*** en ***td*** y ***th***.
- Los atributos ***abbr*** y ***scope*** en ***td***.
- El atributo ***summary*** en ***table***.

Además, HTML no tiene ninguno de los atributos de presentación que estaban presentes en HTML4 ya que sus funciones se manejan mejor con CSS:

- El atributo ***align*** en ***caption***, ***iframe***, ***img***, ***input***, ***object***, ***legend***, ***table***, ***hr***, ***div***, ***h1***, ***h2***, ***h3***, ***h4***, ***h5***, ***h6***, ***p***, ***col***, ***colgroup***, ***tbody***, ***td***, ***tfoot***, ***th***, ***thead*** y ***tr***.
- Los atributos ***alink***, ***link***, ***text*** y ***vlink*** en ***body***.
- El atributo ***background*** en ***body***.
- El atributo ***bgcolor*** en ***table***, ***tr***, ***td***, ***th*** y ***body***.
- El atributo ***border*** en ***object***.
- Los atributos ***cellpadding*** y ***cellspacing*** en ***table***.
- Los atributos ***char*** y ***charoff*** en ***col***, ***colgroup***, ***tbody***, ***td***, ***tfoot***, ***th***, ***thead*** y ***tr***.
- El atributo ***clear*** en ***br***.
- El atributo ***compact*** en ***dl***, ***menu***, ***ol*** y ***ul***.
- El atributo ***frame*** en ***table***.
- El atributo ***frameborder*** en ***iframe***.
- El atributo ***height*** en ***td*** y ***th***.
- Los atributos ***hspace*** y ***vspace*** en ***img*** y ***object***.
- Los atributos ***marginheight*** y ***marginwidth*** en ***iframe***.
- El atributo ***noshade*** en ***hr***.
- El atributo ***nowrap*** en ***td*** y ***th***.
- El atributo ***rules*** en ***table***.

- El atributo ***scrolling*** en ***iframe***.
- El atributo ***size*** en ***hr***.
- El atributo ***type*** en ***li*** y ***ul***.
- El atributo ***valign*** en ***col***, ***colgroup***, ***tbody***, ***td***, ***tfoot***, ***th***, ***thead*** y ***tr***.
- El atributo ***width*** en ***hr***, ***table***, ***td***, ***th***, ***col***, ***colgroup*** y ***pre***.

Hasta aquí, hemos recogido la lista completa de novedades en lo referente a los atributos y hemos ilustrado el funcionamiento de los más interesantes, o los que parece que están teniendo o pueden tener mayor repercusión.

Tenga en cuenta el lector que, en algunos casos, el soporte de los navegadores no está garantizado y que existen propuestas dentro de este conjunto de novedades, que –al igual que sucede con los elementos- es posible que desaparezcan de la especificación por falta de soporte en los navegadores.

El próximo capítulo, lo dedicaremos en su integridad a otro de los componentes del estándar que ha tenido mayor número de cambios: las hojas de estilo en cascada versión 3 (CSS3).

Referencias

- Atributos cambiados en la especificación HTML5: http://www.w3.org/TR/2013/WD-html5-diff-20130528/#changed-attributes
- "*Introducción a WAI-ARIA*", en http://web.archive.org/web/20100622122838/http://www.areia.info/introduccion-a-wai-aria
- "*Web Accessibility and WAI-ARIA Primer*", Emily P. Lewis, en MSDN Magazine: http://msdn.microsoft.com/en-us/magazine/ff743762.aspx
- "*Plantillas de proyecto JavaScript para aplicaciones estilo Metro*", MSDN, http://msdn.microsoft.com/es-es/library/windows/apps/hh758331.aspx

Capítulo 04 | Los estándares de CSS 3

Definición y objetivos

La inefable Wikipedia, define las Hojas de Estilo en Cascada diciendo: "*Las hojas de estilo en cascada o (Cascading Style Sheets, o sus siglas CSS) hacen referencia a un lenguaje de hojas de estilo usado para describir la presentación semántica (el aspecto y formato) de un documento escrito en lenguaje de marcas. Su aplicación más común es dar estilo a páginas webs escritas en lenguaje HTML y XHTML, pero también puede ser aplicado a cualquier tipo de documentos XML, incluyendo SVG y XUL.*

La información de estilo puede ser adjuntada como un documento separado o en el mismo documento HTML. En este último caso podrían

definirse estilos generales en la cabecera del documento o en cada etiqueta particular mediante el atributo "<style>".

El problema con esta definición es que utiliza la palabra lenguaje, cuando, más bien, es una sintaxis: una forma de definir reglas de presentación que serán usadas más tarde por un intérprete que forma parte un agente de usuario (pongamos un navegador) para decidir cómo se muestran visualmente los elementos de una página Web.

El primer objetivo y el más importante, se consiguió con su mera existencia: Desde la especificación 1.0 de CSS se conseguía separar el contenido de la presentación, algo que ahora nos parece habitual en muchas soluciones informáticas pero que 10 años atrás no lo era tanto.

Estandarización

CSS 1.0 ha sido sobrescrito y mejorado hasta la versión actual, y como ya se ha dicho, también es la W3C[46]la encargada de la formalización de los documentos que constituyen un estándar que ha sufrido un número de modificaciones y evoluciones en el tiempo, pasando por las versiones 1.0, 2.0, 2.1 (la última completada)[47] y coincidiendo con esta nueva edición, la 3.0.

La novedad de esta versión en cuanto al objetivo planteado al construirla es que todos los creadores de agentes de usuario estaban ansiosos de poder implementar algo lo antes posible. Solución: ir creando módulos más pequeños de funcionalidad, e irlos publicando paulatinamente, de forma que pudieran ser implementados con las sucesivas actualizaciones, específicos de un área concreta.

Cada módulo, añade nuevas capacidades o extiende las existentes en la especificación CSS 2.0/2.1. Debido a esto, módulos diferentes poseen diferente status y diferente nivel de estabilidad e implementación en los

[46] http://www.w3.org/Style/CSS/

[47] http://www.w3.org/TR/CSS2/

agentes de usuario o navegadores.

Nos encontramos, por tanto con que módulos, como "**Selectors**", "**Namespaces**", "**Color**", "**Media Queries**", "**Backgrounds and Borders**" y "**Multicolumn Layout**" se consideran ya en estado completado o estable, mientras otras caminan con algo más de retraso. Aun así, la W3C está haciendo todo lo posible por igualar el proceso de avance de forma que se llegue a un nivel adecuado en un rango manejable de fechas.

La figura 1 muestra –literalmente- el fragmento inicial de la tabla de especificaciones del estándar tal y como se encuentra a primeros de agosto de 20148. Dada la naturaleza dinámica de este documento, recomendamos que se consulte la página siempre que se desee ver la situación de un determinado módulo de CSS3.

La simbología utilizada por el sitio de publicación respecto a cada módulo es la siguiente:

[48] http://www.w3.org/Style/CSS/current-work

TABLE OF SPECIFICATIONS

Ordered from most to least stable:

Completed	Current	Upcoming	Notes
CSS Snapshot 2010	NOTE	–	Latest stable CSS
CSS Snapshot 2007	NOTE	–	
CSS Color Level 3	REC	REC	See Errata
CSS Namespaces	REC	REC	
Selectors Level 3	REC	REC	
CSS Level 2 Revision 1	REC	REC	See Errata
CSS Level 1	REC	–	Unmaintained, see Snapshot
CSS Print Profile	NOTE	–	
Media Queries	REC	REC	

Stable	Current	Upcoming	Notes
CSS Style Attributes	CR	PR	

Testing	Current	Upcoming	Notes
CSS Backgrounds and Borders Level 3	CR	LC	
CSS Conditional Rules Level 3	CR	CR	
CSS Image Values and Replaced Content Level 3	CR	PR	
CSS Marquee	CR	PR	
CSS Multi-column Layout	CR	CR	
CSS Speech	CR	PR	
CSS Values and Units Level 3	CR	PR	
CSS Flexible Box Layout	CR	PR	
CSS Text Decoration Module Level 3	CR	PR	
CSS Mobile Profile 2.0	CR	PR	Status unknown
CSS TV Profile 1.0	CR	?	Status unknown

Fig. 1: Estado actual de los documentos de la especificación CSS3

Y el significado de las siglas que se utilizan en estos documentos, viene explicado con más detalle en la Tabla 1.

Abreviatura	Nombre completo
WD	***Working Draft*** (Borrador)
LC	***Last Call*** (Última llamada)
CR	***Candidate Recommendation*** (Rec. Candidata)
PR	***Proposed Recommendation*** (Rec. Propuesta)
REC	***Recommendation*** (Recomendación)

Tabla 1: Significados de cada marca de la especificación

Afortunadamente, no se precisa que todas las especificaciones estén concluidas para comenzar a trabajar en las partes principales. El nivel de implantación de CSS, es –al igual que sucede con HTML 5- desigual, pero todos los fabricantes están trabajando muy activamente para incluir más soporte en cada nueva versión que aparece de un navegador.

Soporte del estándar CSS 3 en Visual Studio 2012/2013

En el caso de IE 10/11, el soporte es muy completo, y especialmente, en las últimas versiones de Visual Studio, se ha extendido a todos los aspectos del desarrollo: Editores, Visualización en *Page Inspector*, soporte *Intellisense*, etc.

En la edición de código, se ha incluido soporte especial para aspectos como el color, añadiendo una caja de diálogo contextual que aparece en el momento de la edición de cualquier propiedad relacionada con él, que incorpora un selector dinámico de colores para capturar cualquier tono de color disponible en el escritorio en ese momento, como vemos en la figura 2:

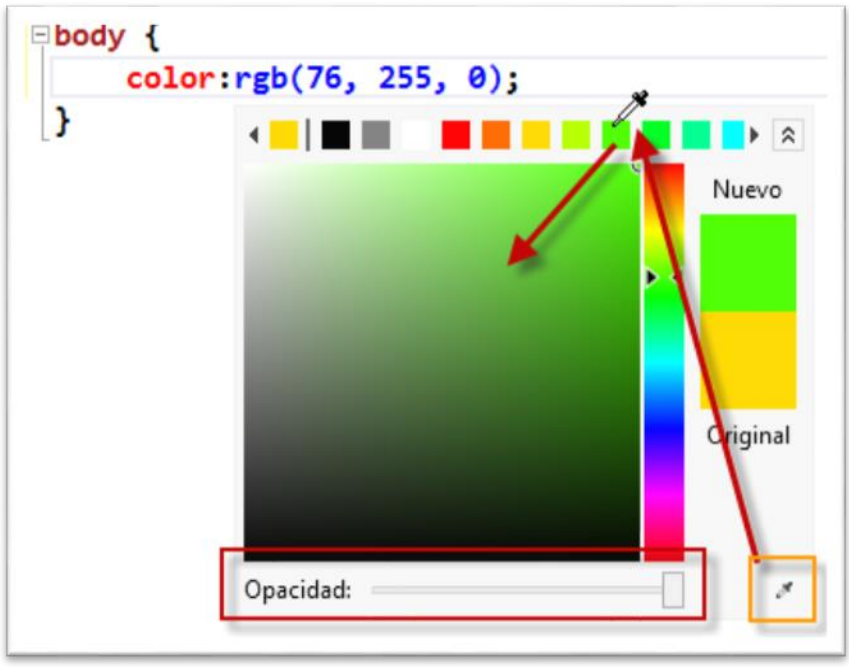


Fig. 2: El Editor CSS y la selección de colores en Visual Studio 2012/2013.

En lo relativo a las palabras reservadas propias de CSS 3, el soporte es igualmente muy completo, incluyendo todas las extensiones para los diversos navegadores, como podemos apreciar en la figura 3:

Fig. 3: Soporte Intellisense de CSS3 en Visual Studio 2012/2013

Otro aspecto fundamental es la depuración. Hasta ahora, la depuración se integraba con el navegador seleccionado para ver la página, pero con esta versión, disponemos del nuevo ***Page Inspector***, que nos permite manejar la depuración controlando los 3 lenguajes implicados en la ejecución de las páginas Web: HTML, CSS y JavaScript.

Depuración visual con *Page Inspector*

Por centrarnos en el apartado de CSS, **Page Inspector** permite –en tiempo de ejecución- marcar cualquier elemento de la interfaz de usuario, y filtrar exclusivamente aquellos aspectos aplicables a ese elemento. Nos muestra automáticamente una ventana con varias solapas donde podemos ver, respectivamente, los estilos aplicables a ese elemento, dónde se utiliza un estilo concreto, cual es la disposición visual de los elementos e, incluso, nos permite simular cuál sería la apariencia del elemento si le aplicamos cualquier cambio o le añadimos un nuevo atributo.

En un sencillo código de prueba para comenzar, hemos utilizado un texto tipo "*Lorem Ipsum*"[49] distribuido en varios párrafos (elementos **<*p*>**), a cuya página contenedora hemos asociado una hoja de estilo separada, arrastrándola desde el Explorador de Soluciones a la superficie del editor de CSS, lo que hace que Visual Studio 2012 nos genere una instrucción como esta:

```
<link rel="stylesheet" type="text/css"
href="CSS/Hoja1.css" />
```

El aspecto del código de los diversos párrafos es similar al siguiente (observe que no incluimos ningún tipo de atributo en los elementos HTML; solo usamos texto plano):

```
<p>Nam magna urna, hendrerit a laoreet et, tincidunt eget
massa. Nulla at ligula nibh, vitae porta metus. Nulla
ulputate, urna quis rutrum consequat, est dui venenatis
```

[49] http://es.lipsum.com/

```
elit, id sollicitudin nunc dolor nec mauris. Nam ac mi in
lorem dictum viverra eu sed arcu. In suscipit
sollicitudin venenatis. Nunc sit amet erat sit amet dui
laoreet ullamcorper a vel metus. Donec lorem felis,
eleifend ut ultrices ut, bibendum nec sapien. Etiam
mollis fringilla risus id pharetra. Aenean ac libero leo,
nec dignissim quam.</p>
```

En ejecución, seleccionamos *Page Inspector* como navegador predeterminado, y podemos marcar cualquier elemento del código fuente: la solapa "*Styles*", nos mostrará qué estilos están siendo aplicados a ese elemento en ese instante, como podemos apreciar en la Figura 4:

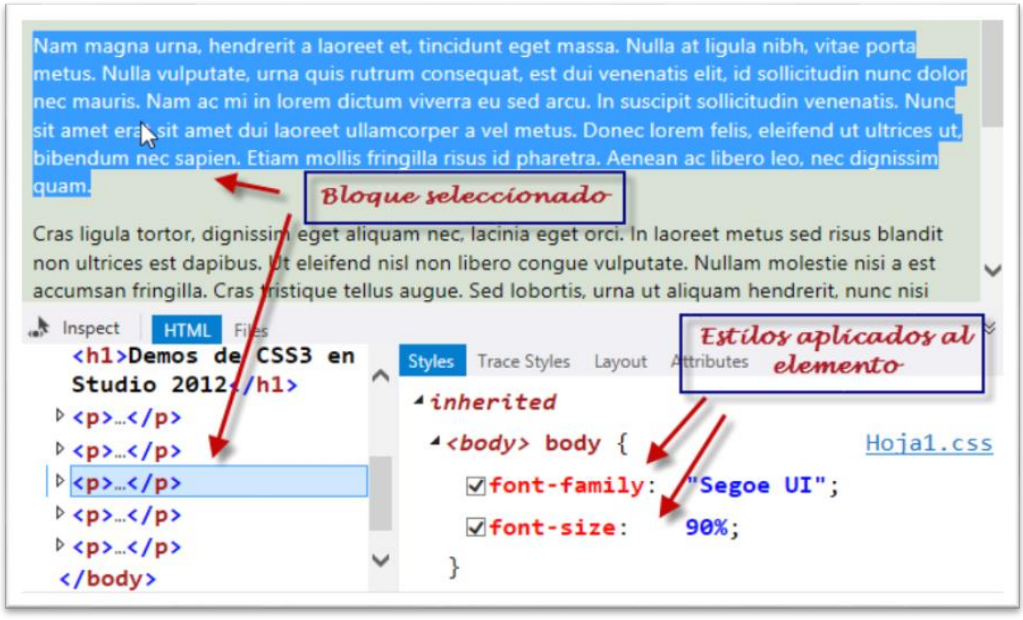


Fig. 4: Page Inspector mostrando el soporte de depuración para CSS3 de V. Studio 2012

De la misma forma, si tenemos varios estilos que pueden aplicarse a un mismo elemento, podemos abordar el problema a la inversa, es decir, podemos examinar a qué elemento se aplica un estilo concreto.

Por ejemplo, creamos una clase CSS llamada "*Parrafo*" para aplicar a

algunos elementos **<p>** de nuestro código, y la usamos en nuestro ejemplo anterior. Una vez seleccionada, podremos inspeccionar la ventana "*Trace Styles*", y veremos (Fig. 5), el "histórico" de utilización de ese estilo. En la imagen, vemos cómo el estilo "*font-family*", definido para la clase "*Parrafo*", es el que se está usando, aunque existe una definición anterior (para el elemento **<body>**) que aparece tachada en la imagen:

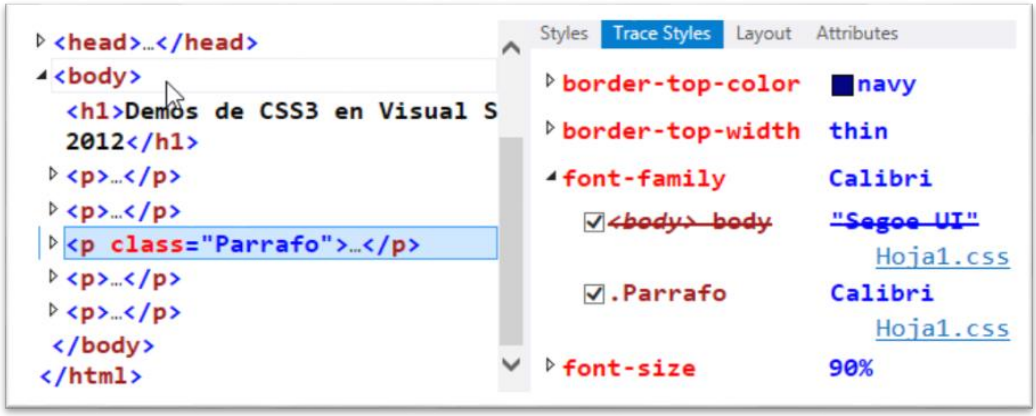


Fig. 5: La solapa "Trace Styles" dentro de Page Inspector

Los *checkbox* que aparecen en la ventana pueden desmarcarse, haciendo que deje de aplicarse el estilo definido y viendo el resultado directamente en la ventana de ejecución. También podemos añadir estilos y reglas dinámicamente y comprobar directamente el resultado.

A lo largo de este capítulo podremos apreciar más aspectos interesantes del soporte que el nuevo IDE ofrece respecto al estándar CSS3.

Las ventajas de CSS

Independientemente de la versión, el uso de CSS tanto para sitios como para aplicaciones Web, es, hoy día, la forma recomendada para definir el aspecto visual en HTML. Conviene recordar las razones de más peso:

- **Flexibilidad**. El propio principio de la separación de responsabilidades (*separation of concerns*), se manifiesta aquí desde el comienzo. Los cambios en las paletas de diseño, los tipos de letras

asociados con zonas y fragmentos de las aplicaciones, el tipo de formato, la imagen corporativa, etc., se manejan mucho más fácilmente al estar separados funcional y físicamente, de forma que no se precisa tocar una página para realizar cambios (incluso cambios muy significativos), si el diseño ha partido de estas consideraciones desde el inicio.

- **Programación**. El soporte del DOM y algunos de los nuevos atributos y API que ya hemos visto o veremos en el apartado de JavaScript, permiten la creación, modificación, reasignación y manipulación dinámicas de los elementos CSS vinculados a cualquier elemento. Eso permite otro grado más de flexibilidad a los comentados en el punto anterior.
- **Accesibilidad**. La posibilidad de acceder y modificar el conjunto de nuevos atributos WAI-ARIA desde CSS, permite crear páginas con distintos grados de accesibilidad, seleccionando dinámicamente la hoja de estilo que más se ajuste al nivel requerido por el lector.
- **Personalización**. Por la misma razón, es posible combinar las posibilidades ofrecidas por las *cookies* que guarden preferencias de usuario, y establecer modificaciones iniciales, ofreciendo un aspecto individualizado, o incluso habilitando la presencia o ausencia de ciertos elementos en función de esas preferencias.
- **Consistencia entre sitios**. Una vez definida y probada una hoja de estilo podemos reutilizarla en multitud de sitios sin miedo a cambios en el comportamiento. Para ello debemos realizar diseños auto-contenidos, en pequeñas unidades bien probadas, que puedan ser extrapoladas a otros contextos más adelante, con expectativas de un funcionamiento similar.
- **Economía del ancho de banda**. El uso de hojas de estilo, permite reducir muchísimo la verbosidad necesaria para expresar lo mismo mediante etiquetas HTML. Además, éstas deben centrarse en el contenido, su contexto, sus aspectos semánticos y su estructura como documento organizado, y no en los aspectos visuales asociados.

En una interesante descripción, **Jason Crawford Teague**, en su obra "*CSS 3. Visual QuickStart Guide*" (ver referencias al final del capítulo), define CSS junto a los otros 2 protagonistas de la Web de forma similar a las

partes de un discurso. HTML proporciona los nombres, CSS los adjetivos y los adverbios, y JavaScript los verbos. Esta filosofía se ha hecho aún más patente con la versión CSS 3.

Ubicación de los estilos y ámbito de influencia

Como parte de este recordatorio de CSS, tengamos presente que la ubicación de una definición de estilo depende del ámbito de influencia que nos interese. Un estilo puede incorporarse a cualquier elemento, asociándolo con su atributo ***style***. Aunque no es la mejor práctica, esto tiene sentido cuando se trata de alguna configuración única que no se va a repetir en ninguna otra parte de la página o el sitio. En este caso, el ámbito se reduce exclusivamente al elemento.

El segundo caso se presenta con un ámbito local a la página. Si, por las razones que sean, es más conveniente disponer de ciertas definiciones exclusivas de una página, podemos definir el estilo utilizando la etiqueta HTML correspondiente (**<*style*>**). Una etiqueta de este tipo puede estar ubicada dentro de la sección **<*head*>** o de la sección **<*body*>**, y puede aparecer tantas veces como sea necesario en la misma página.

Sin embargo, la ubicación habitual es en ficheros separados. Un fichero de Hojas de Estilo en Cascada, suele tener la extensión .CSS (aunque esto no es imprescindible), y debe ser válido según la sintaxis del estándar. En caso de que se produzcan errores sintácticos, cualquier atributo definido pero no comprendido por el navegador no produce un error: simplemente, se ignora.

En este caso, para establecer la relación entre la página y las definiciones de estilos haremos referencia a ésta utilizando la etiqueta **<*link*>**, con una sintaxis como la siguiente:

```
   <link rel="stylesheet" type="text/css"
href="Hoja1.css" />
```

(Recordemos que Visual Studio nos permite arrastrar el fichero de estilos desde el Explorador de Soluciones directamente a zona de la página HTML que deseemos).

El concepto de Selector

Quién ya conozca CSS, puede saltar este apartado, pues vamos a recordar el concepto de selector y los tipos de selectores existentes en CSS, e incluiremos al final los que son nuevos en esta versión.

Un selector es un mecanismo por el cual establecemos –sin ambigüedades- cuál es el destino de las directivas de diseño que estamos definiendo (a qué elementos se deben aplicar las reglas CSS).

Se definen mediante 3 recursos sintácticos: los *Selectores HMTL*, las *Clases* y los *Identificadores*. Describimos brevemente, su funcionalidad.

Formato

Cualquier tipo de selector, sigue el patrón sintáctico siguiente:

Selector { Propiedad : Valor }

Donde *Selector* puede referirse a una etiqueta, a una clase o a un identificador. Las reglas de sintaxis de los selectores son sencillas:

- Usar solamente letras y números, guion bajo o guiones medios.
- El primer carácter no puede ser ni numérico ni comenzar con un guion.
- No utilizar espacios.

Los selectores HTML son aquellos que se corresponden con el texto de su etiqueta. Por ejemplo, la etiqueta ****** tiene como selector ***span***. Y la expresión en que se incluye lo muestra seguida de la indicación a aplicar expresada entre corchetes, formado por un conjunto de pares clave/valor:

```
span { font-size: 21px; }
```

Las **clases** son palabras reservadas definidas por el usuario para indicar un conjunto de reglas aplicables a los elementos que –más adelante- se consideren convenientes, consiguiendo así una gran versatilidad. Se definen utilizando una palabra válida precedida por un punto. Por

ejemplo:

```
.Parrafo  { color: blue; }
```

Y se utilizan en cualquier elemento indicando ese nombre (sin el punto), en el atributo ***class*** correspondiente:

```
<h2 class="Parrafo">Saludos</h2>
```

El uso del código anterior provocara que el texto "Saludos" aparezca en color azul.

Los identificadores funcionan de forma parecida a los selectores, solo que se aplican al elemento definido por ese identificador, indicando el nombre precedido por el símbolo de # (almohadilla). Por ejemplo, en el código anterior, si identificamos un elemento ***<h2>*** de la siguiente forma:

```
<h2 Id="Saludo">Saludos</h2>
```

…podremos obtener un resultado equivalente al anterior con el siguiente código CSS:

```
#Saludo { color: blue; }
```

De hecho, las 3 características pueden combinarse para permitir un ajuste más fino a la hora de seleccionar a qué elementos es aplicable una configuración CSS concreta.

Valores enumerados

Siguiendo lo definido por versiones anteriores de CSS, es posible igualmente la enumeración de valores, mediante una lista separada por comas, en aquellas propiedades en las que tenga sentido. Por ejemplo, podemos indicar varias alternativas para un tipo de letra enumerando los nombres de las fuentes de la siguiente forma:

```
font-family: Arial, Tahoma, 'Segoe UI', 'Times New Roman';
```

El navegador tratará de localizarlas en el orden indicado, y, si no encontrase ninguna de ellas en el Sistema, utilizaría la fuente definida de manera predeterminada.

(Observe que las Fuentes cuyo nombre consta de más de una palabra se expresan entre comillas simples).

Los navegadores y las Extensiones CSS

Muchos navegadores optaron por "completar" la especificación con implementaciones adicionales que ofrecían otra funcionalidad que –si bien no es parte de la especificación- aumenta el abanico de opciones y soluciona algunas carencias.

Esto supone que el código puede no ser totalmente portable entre distintos agentes, De ahí que hayan aparecido lo que se denominan "Extensiones CSS de navegador". Estas extensiones se reconocen mediante un prefijo propio para cada navegador, tal y como vemos en la tabla siguiente:

Extensión	Motor	Navegador(es)	Ejemplo
-moz-	Mozilla	Firefox	`-moz-appearance:checkbox;`
-ms-	Trident	I. Explorer	`-ms-grid-layer:inherit;`
-o-	Presto	Opera	`-o-tab-size:initial;`
-webkit-	Webkit	Chrome, Safari	`-webkit-transition:inherit;`

Tabla 2: Extensiones CSS de los navegadores

Donde vemos las extensiones de los cinco navegadores que analizamos aquí, junto al motor de *rendering* utilizado, y un pequeño ejemplo de uso.

> **Nota**: Visual Studio 2012/13 dispone de soporte de todas estas extensiones cuando creamos un fichero de este tipo. Además, no hay problema para incluir cualquiera de estas extensiones en nuestro código, ya que los navegadores sólo utilizan las que les son propias.

Selectores combinados (o dependientes)

Un selector combinado viene definido en función de otro selector ya existente. Por ejemplo, si queremos crear una variante de la clase anterior "***Parrafo***", que se aplique solo a los elementos ***span*** que sean de esa clase, podemos usar el código siguiente:

```
span.Parrafo { color: Green; }
```

De la misma forma, podemos aplicar varias clases simultáneamente a un elemento HTML indicándolo en el atributo ***class***, separando los nombres de las clases con espacios.

El selector universal (nuevo en CSS3)

En CSS 3, existe un nuevo tipo de selector llamado Selector Universal, que se define por el signo de * (asterisco), y que permite definir estilos que serán utilizados por todos los elementos del contenedor donde esté definida, sin excepción. Resulta útil si nuestra página tiene alguna característica compartida, por ejemplo, por criterios empresariales.

Así pues, si queremos aplicar un margen y una separación igual entre texto y bordes (***padding***) para todos los elementos, podemos definir lo siguiente:

```
* {
    margin: 3px;
    padding: 5px;
  }
```

También podemos utilizarlo para indicar que una regla se aplique a todos los elementos que sean parte de otro mediante la sintaxis de pertenencia que veremos a continuación.

Téngase en cuenta que, aunque la aplicación de los estilos se produce en cascada (lo que quiere decir que un elemento contenedor superior propaga sus propiedades a los elementos contenidos), cada elemento puede contener su propia definición que sobrescriba la de su contenedor. De esta forma se garantiza la globalidad en la aplicación del estilo.

Agrupación de selectores

También es posible utilizar el mismo estilo para diversos selectores, agrupándolos en la definición, y separándolos por comas, antes de la aparición del primer corchete. Por ejemplo, si queremos que la definición anterior se aplique a varios elementos, una clase y un identificador indistintamente, podríamos escribir lo siguiente:

```
span, h1, .Parrafo, #Saludo { font-size: 45px; }
```

De la misma forma, podemos establecer agrupaciones de selectores basándonos en la posición de los elementos dentro del árbol HTML (el DOM). Por ejemplo podemos establecer cómo tiene que aparecer un elemento en función de sus contenedores o en otros elementos de su mismo nivel (*sibligs*).

Un ejemplo: podemos indicar una excepción en el color de la letra de un elemento dado, si éste utiliza una clase determinada con la siguiente sintaxis:

```
.Parrafo span { color:red; }
```

Esto indica que los elementos ****** que sean descendientes de otro cuyo atributo ***class*** sea "*Parrafo*" irán en rojo. Observe que en este caso, la enumeración no va separada por comas, sino por un espacio.

De la misma forma, podríamos combinar esta sintaxis con la de los selectores universales para indicar colecciones más restringidas, como por ejemplo:

```
.Parrafo * { color:red }
```

Que indica que cualquier elemento (del tipo que sea, no solo <***span***>) contenido en otro cuyo atributo ***class*** sea "*Parrafo*", irá en rojo. O sea, la universalidad de *******, queda, en este caso, limitada a los elementos de clase "*Parrafo*".

Otros mecanismos de selección

La potencia de los selectores no sería tan evidente si no dispusiésemos de otras formas adicionales y combinadas de indicar una regla a cualquier elemento. Concretamente, podemos seleccionar los elementos por 3 criterios distintos:

- Selección por las relaciones entre los elementos
- Selección por el valor de sus atributos
- Selección por la interacción con el usuario

Una nota sobre las pseudo-clases

Antes de continuar, es importante tener claro el concepto de pseudo-clase. En CSS, se denominan **pseudo-clases** a palabras reservadas que adoptan la forma

elemento : pseudo-clase

Permiten actuar sobre los elementos, dependiendo de cierta información que, de manera predeterminada, se encuentran fuera del DOM, o tiene que ver con su concepto de estado (*visitado*, *marcado*, *activo*, deshabilitado, etc.), o si están vacíos, o por la posición que ocupan respecto a otros elementos.

El ejemplo canónico es el del elemento **<*a*>**, que tiene los estados

normal (***link***), visitado (***visited***), cursor encima (***hover***) y pulsado (***active***). Cada uno de esos estados podemos definirlo y programarlo independientemente y esa es una función de las pseudo-clases.

Otros elementos pueden tener varios estados como ***<iput type=checkbox>*** que puede encontrarse en estado *marcado, no marcado* e *indeterminado* y también podemos encontrarnos con elementos en los que la presencia de un atributo, cambia su estado, como es el caso del atributo ***required*** que hemos visto en el capítulo anterior.

Pseudo-clases por posición

El estándar CSS3 define las siguiente pseudo-clases por la posición que ocupan los elementos en relación con otros (***e*** hace referencia al elemento):

- ***e*:root** Elemento raíz
- ***e*:nth-child()** n-ésimo descendiente
- ***e*:nth-of-type()** n-ésimo de un tipo dado
- ***e*:nth-last-of-type()** n-ésimo de un tipo contando desde el final
- ***e*:first-of-type** Primero de un tipo dado
- ***e*:last-of-type** Último del tipo…e
- ***e*:only-of-type** Único del tipo…e
- ***e*:only-child** Único descendiente
- ***e*:last-child** Último descendiente

Estos pseudo-elementos, como podemos ver, lo que definen realmente son posiciones relativas al contexto. Debemos ser cuidadosos al utilizarlas puesto que la adición (o eliminación) de un elemento a una colección puede suponer que, al aplicar la regla nuestro elemento de destino original deje de estar seleccionado por ella.

Pseudo-clases por otros criterios

Otros criterios posibles de selección no tienen que ver con la posición relativa, sino con otros aspectos, como si se encuentra vacío, habilitado

o marcado, como vemos en la lista siguiente:

- **e:not()** **Pseudo-operador de negación (condiciones lógicas)**
- **e:target** **Destino de una URI**
- **e:empty** **Vacío**
- **e:enabled** **Habilitado**
- **e:disabled** **Deshabilitado**
- **e:checked** **Verificado**
- **e:lang** **Lenguaje (de un tipo dado)**
- **e:indeterminate Indeterminado (controles tipo *bool* sin valor).**
- **e:link** **Enlace no visitado**
- **e:visited** **Enlace visitado**
- **e:active** **El elemento están tomando parte en acciones del usuario**
- **e:focus** **El elemento tiene el foco**
- **e:required** **El elemento tiene el atributo *required***
- **e:optional** **El contenido del elemento es opcional**
- **e:valid** **El tipo de dato y el contenido coinciden.**
- **e:invalid** **El elemento es de un tipo de dato y el contenido no coincide.**
- **e:in-range:** **El valor del elemento se encuentra dentro del intervalo indicado por sus atributos min y max.**
- **e:out-of-range: El valor del elemento se encuentra fuera del intervalo indicado por sus atributos min y max.**

Todos ellos, son un destino posible de un selector, como vamos a ver a continuación, si bien es cierto que los dos últimos valores de la lista, por ejemplo, no están aún soportados por los navegadores.

Sobre estas bases, el nuevo estándar extiende estas ideas añadiendo un conjunto completo de pseudo-clases y cambios sutiles en la sintaxis para diferenciar los pseudo-elementos.

El ejemplo habitual que nos encontramos consiste en modificar el comportamiento determinado de los enlaces de un sitio para los diferentes estados, como muestra el siguiente código:

```
a:hover
{
    text-decoration: none;
}

a:visited
{
    color:green;
}
```

En este caso todos los enlaces visitados se presentarán en verde, y mientras el usuario pasa el cursor por encima de un enlace, el subrayado típico no se mostrará.

La potencia obtenida con estas opciones es muy grande, así como sus usos inmediatos, como puede ser el caso de la pseudo-clase ***:empty*** que nos permite configurar la aparición de elementos existentes, pero vacíos. Este caso es especialmente útil en aquellas situaciones en las que por razones de enlace de datos a un origen (*databinding*), uno de los elementos puede llegar a tomar el valor ***null***, lo que puede traducirse como la cadena vacía o una situación similar.

De la misma forma, el lector puede probar situaciones aplicables a otras pseudo-clases interesantes, como ***:target***, ***:disabled***, ***:checked*** o ***:root.***

> Nota: la diferencia entre el pseudo-elemento ***:root*** y un selector para el elemento HTML es doble: por un lado, ***:root*** tiene prioridad, por otro, éste puede aplicarse a otros documentos, como los XML, cuya raíz es diferente.

Lo mismo puede afirmarse del pseudo-operador de negación que nos permite programar circunstancias excepcionales, como por ejemplo, "todos los **<p>** que no utilicen la clase **.Parrafo**", como podría ser el siguiente ejemplo:

```
p:not(.Parrafo)
{
color: red;
font-style: italic;
}
```

Selección por las relaciones entre los elementos

Una vez visto el concepto de pseudo-clase, podemos categorizar los selectores. En CSS (y más en esta versión), permite seleccionar un elemento por la posición que ocupa en el DOM (Modelo de Objetos del Documento), ya sea de forma absoluta o relativa a otros elementos.

De esta forma, podemos seleccionar un elemento dependiendo de quiénes son sus descendientes (directos o de cualquier nivel de profundidad), de quién es su antecesor (o elemento contenedor), de qué elementos lo rodeen, y así sucesivamente.

La tabla siguiente muestra una tabla básica de relaciones que ya funcionaba en las versiones anteriores del estándar, indicando la sintaxis que se sigue en cada una de las relaciones posibles:

Formato	Selector	Condición
`a b c`	Descendiente	**c** descendiente de **b** descendiente de **a**
`a * b`	Universal	**b** dentro de **a** para cualquier ancestro de **b**
`a > b`	Hijo directo	**b** es hijo directo de **a**
`a + b`	Adyacente de mismo nivel	**b** es adyacente a **a** y de su mismo nivel
`a ~ b`	Mismo nivel	**b** es del mismo nivel que **a**

Tabla 3: Combinaciones de Selectores

Por ejemplo, con la siguiente sintaxis:

```
.Parrafo > div > p { color:green }
```

Estamos indicando que todos los elementos ***<p>*** que sean hijos directos de un elemento *** *** cuyo ancestro tenga el atributo ***class*** con el valor "*Parrafo*", irán en color verde.

De la misma forma, podemos utilizar cualquiera de los pseudo-elementos de posición listados al principio para establecer el criterio de selección. Debemos tener en cuenta que en la estructura del DOM no existen mecanismos que nos indiquen o permitan seleccionar elementos por su orden numérico o su posición en el árbol de elementos –por ejemplo, no podemos saber cuál es el tercer párrafo (elemento **<p>**) desde el comienzo de una etiqueta *** ***, salvo que utilicemos esta opción.

De esta forma, si contamos con una estructura como la siguiente:

```
<div>
    <p>Párrafo 1</p>
    <p>Párrafo 2</p>
    <p>Párrafo 3</p>
    <p>Párrafo 4</p>
</div>

<div>
    <p>Párrafo 5</p>
    <p>Párrafo 6</p>
    <p>Párrafo 7</p>
    <p>Párrafo 8</p>
</div>
```

Podemos definir una modificación de todos los terceros párrafos contenidos en cualquier otro elemento con esta sintaxis:

```
p:nth-of-type(3)
{
    font-size:18px;
```

```
    color: Red;
}
```

El tamaño de fuente y el color seleccionados sólo se mostrarían para aquellos párrafos que –estando contenidos en otro elemento- ocupen la tercera posición de los de su tipo, como vemos en la figura adjunta:

Fig. 6: Salida en IE10/Page Inspector del código anterior

Selección por el valor de sus atributos

Como una extensión de lo anterior, en CSS 2 se introdujo la noción de selectores por valores de un atributo. En esa especificación, solo existían 4 selectores disponibles que vemos en la siguiente lista:

```
Elemento [attr] {}          /* Selector de presencia de
atributo */
Elemento [attr='value']{} /* Selector de comparación
atributo/valor*/
```

```
Elemento [attr~='value']{} /* Selector de comparación
parcial */
Elemento [attr|='value']{} /* Selector de atributo por
lenguaje */
```

En el primer caso, se seleccionan los elementos que posean el atributo, ya tengan o no un valor asignado. Por ejemplo imaginemos que tenemos las siguientes 4 etiquetas como parte de un menú:

```
<a href="Page1.html" rel="Value11"   lang="en-GB">Link
1</a>
<a href="Page2.html" rel="Value11 2" lang="en-US">Link
2</a>
<a href="Page3.html" rel="Value13"   lang="en-AU">Link
3</a>
<a href="Page4.html" rel="Value14"   lang="es-ES">Link
4</a>
```

El siguiente código CSS selecciona los 4 elementos ***<link>*** y los pone todos en rojo:

```
a[rel] { color: red; }
```

El segundo selecciona los que, teniendo el atributo, coincidan con el valor "*Value11*". Si, a continuación del código anterior, añadimos otro carácter, solamente el primer ***<link>*** aparecerá en color verde:

```
a[rel='Value1'] { color: green; }
```

El tercero selecciona los elementos cuyo valor coincida parcialmente con la cadena indicada, entendiendo por tal que empiecen por la misma parte de la cadena, aunque luego haya espacios. Por ejemplo, el siguiente código selecciona los dos primeros enlaces (Link 1 y Link 2):

```
 a[rel~='Value11'] { color: #b200ff; }
```

Y el cuarto, es aplicable a aquellos elementos que indiquen su atributo

Lang (lenguaje). De forma que, para seleccionar el que tiene lenguaje español, escribiríamos lo siguiente:

```
a[lang|='es'] { color: #cdb281; }
```

Como vemos, el lenguaje se ha seleccionado de forma parcial, por lo que afectaría a todos los elementos cuyo lenguaje fuera una variante del español (es-*), como "es-CO" para el caso de Colombia.

Omitimos las salidas de estos ejemplos por ser bastante evidentes, habiéndoles probado en todos los navegadores de forma satisfactoria.

Nuevos selectores de atributos en CSS 3

Con la idea de ofrecer incluso más flexibilidad, en CSS 3 se han añadido nuevas formas de selección que permiten un ajuste aún más fino, y que son igualmente útiles en documentos XML que –por su naturaleza y uso- suelen adoptar valores más diversos.

Se trata de los selectores de *sub-cadena inicial, sub-cadena final, sub-cadena genérica, selección múltiple* y *selector de nivel combinado* (*General Sibling Combinator*).

Los 3 primeros resultan bastante evidentes, y su propósito es de la búsqueda avanzada de sub-cadenas dentro de los valores de un atributo. Dependiendo de la situación, unos u otros parecen más útiles, como es el caso del primero, que permite, por ejemplo, modificar el aspecto de todos los enlaces externos a partir de su atributo ***href***, ya que la inmensa mayoría comenzará por la misma cadena: "*http*".

La sintaxis correspondiente es la que indicamos en la tabla 4:

Tipo de selector	Sintaxis
Sub-cadena inicial	`Elemento [atributo^='valor'] {}`
Sub-cadena final	`Elemento [atributo$='valor'] {}`
Sub-cadena en cualquier posición	`Elemento [atributo*='valor'] {}`

Tabla 4: Tipos de selectores de sub-cadena

No vamos a incluir más ejemplos de estas opciones, que son, prácticamente idénticos a los vistos para los casos anteriores.

Un poco más novedoso es el caso del selector múltiple, que consiste en concatenar distintos selectores simples antes de los corchetes de la definición. Por ejemplo, supongamos que queremos seleccionar todos los enlaces externos que apunten a un fichero PDF para descarga, marcándolos con el color amarillo. El siguiente código cumple la función deseada:

```
a[href^='http://'][href$='.pdf'] { color:yellow; }
```

Obsérvese que el código permite adjuntar consecutivamente los distintos atributos a aplicar en la selección, por lo que podríamos establecer más condiciones adicionales añadiéndolas después de la expresión

```
[href$='.pdf']
```

Y, naturalmente, también se pueden crear todo clase de combinaciones. Precisamente con el objetivo de facilitar esas combinaciones, disponemos ahora del *General Sibling Combinator (GSC)*, que es una extensión del ya existente *Adjacent Sibling Combinator*, (*ASC*) en la versión CSS 2.

En éste último caso, se pueden declarar definiciones que afecten solo a dos elementos adyacentes del mismo nivel con esta sintaxis (se supone a *Elemento1* y *Elemento2* adyacentes y al mismo nivel):

```
Elemento1 + Elemento2 {} /* Adjacent Sibling Combinator
*/
```

Ahora podemos hacer lo mismo con el GSC para todos los elementos que se encuentren en el mismo nivel, con una mínima modificación:

```
Elemento1 ~ Elemento2 {} /* Adjacent Sibling Combinator
*/
```

Aunque muchas de las cosas que pueden hacerse con estas nuevas

opciones ya se podían hacer utilizando la especificación anterior, el objetivo primordial es favorecer la flexibilidad, y permitir más posibilidades, y por tanto la reducción del código necesario.

En lo que respecta al soporte, todas las versiones de los navegadores actuales soportan esta sintaxis de selectores y solo debemos preocuparnos por suministrar ***fallbacks*** en el caso de que tengamos que dar soporte a navegadores antiguos.

Selección por la interacción con el usuario

Como hemos apuntado antes, los elementos de una página no son invariables, sino que, en algunos casos, interactúan con el usuario, y en ciertas situaciones, como consecuencia de ello, cambian su estado.

Ese es el caso del ejemplo que veíamos con el elemento ***a***, que puede tener hasta 4 estos distintos: normal (***link***), visitado (***visited***), cursor encima (***hover***) y pulsado (***active***). Otra situación en la que un elemento muestra un estado es cuando, bien inicialmente o como consecuencia de las interacciones del usuario, un elemento pasa estar deshabilitado (pierde la capacidad de interacción), o la gana pasando de un estado previo inactivo a otro activo.

Además, de los casos anteriores, todos los elementos son capaces de responder (si se les programa de esa forma) al paso del cursor por encima de su superficie, que viene definida por su caja, lo que activa su estado ***hover***, o al abandono de esta zona. Es algo muy habitual utilizar esta característica para la definición de menús dinámicos, por ejemplo.

Menos conocidas son las opciones que ya soportan los navegadores modernos acerca de ciertos elementos que intervienen en operaciones de gestión con el usuario, como por ejemplo aquellos marcados como ***required***. El código siguiente, permite seleccionar los elementos con este atributo en un formulario y marcarlos con un borde especial:

```
<label>
    Requerido
    <input type="text" required>
</label>
<label>
    Opcional
    <input type="text">
</label>
```

Basta con aplicar el siguiente estilo:

```
input:required { outline: 3px solid red; }
```

Para obtener una salida como la que muestra la figura adjunta:

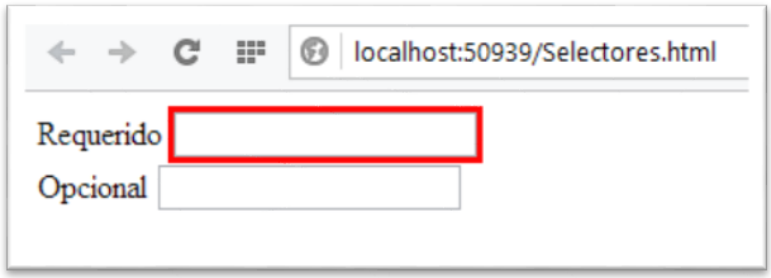


Fig. 7: salida del código anterior

También es novedoso el uso de estas pseudo-clases vinculadas a los elementos input que definen un tipo de dato. Estos elementos admiten los estados ***:valid*** e ***:invalid*** para determinar si el valor introducido puede considerarse correcto según el tipo de ***input*** escogido, como por ejemplo en el caso siguiente con un elemento de tipo ***URL***:

```
<label>
        Valido
        <input type="url"
               value="http://www.danysoft.com">
```

```
</label>
<label>
No válido
      <input type="url" value="Esto no es una URL">
</label>
```

Si aplicamos el estilo `input:invalid { outline: 3px solid red; }` obtendremos una salida como la de la figura siguiente:

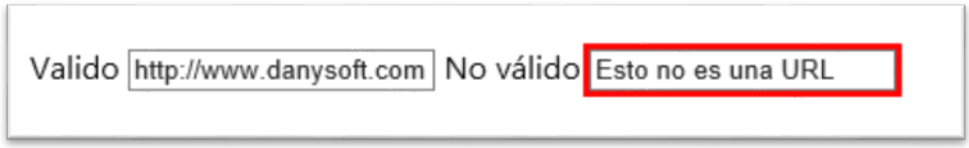


Fig.8: Salida del código para entradas input no válidas

Como vemos, se trata de API que pueden actuar en conjunción con las definiciones de los elementos y que podemos configurar más extensamente via JavaScript en la mayor parte de los casos.

Los pseudo-elementos

Por su parte, los **pseudo-elementos** permiten el acceso a fragmentos de un elemento, como por ejemplo a su primera letra (***first-letter***)[50] o a subconjuntos del contenido, como la primera línea de texto (***first-line***). En el caso de los pseudo-elementos, la sintaxis se duplica (dos símbolos de dos puntos consecutivos (**::**).

La recomendación W3C del 29 de Septiembre de 2011[51], establece como pseudo-elementos los siguientes:

- **::first-letter**
- **::first-line**

[50] Siempre que no vaya precedida de otro elemento

[51] http://www.w3.org/TR/selectors/

- **::before**
- **::after**
- **::selection (Nuevo en CSS 3, pero pendiente de aprobación)**

Y su uso es igualmente, muy sencillo. Por ejemplo, para poner en mayúsculas la primera línea de cualquier párrafo podríamos escribir:

```
p::first-line
{
    text-transform: uppercase;
}
```

Y ambas opciones pueden combinarse, de forma que se consigan efectos más espectaculares sobre el texto. Por ejemplo podemos añadir un tratamiento especial a la primera letra de cada elemento ***<p>*** con un código como el siguiente:

```
p::first-letter
{
    font-family: Aharoni;
    font-size:270%;
}
```

Con lo que obtendríamos una salida similar a la de la figura 9:

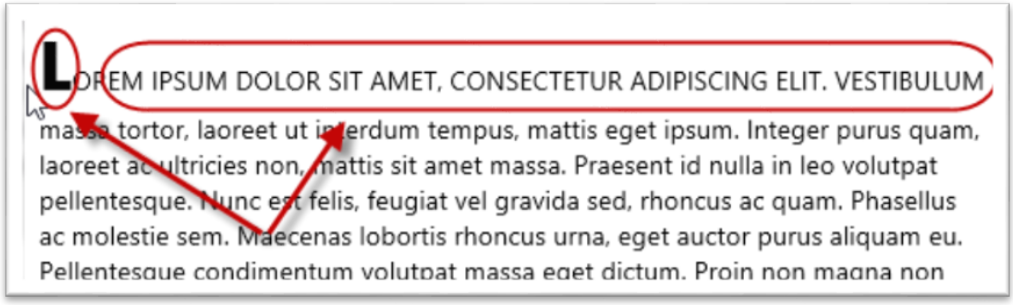


Fig. 9: Resultado de aplicar dos definiciones de pseudo-elementos

Téngase en cuenta, además, que estos principios no son solo aplicables a los elementos estáticos de una página, sino también a los generados

dinámicamente. De hecho, si el usuario modifica la anchura del documento, el número de palabras afectadas por la regla ***::first-line***, se verá afectada.

Los pseudo-elementos *::before* y *::after*

Ya desde la versión CSS 2.1 los desarrolladores hemos dispuesto de mecanismos que nos permiten utilizar varias capas asociadas con un mismo elemento. Este recurso, muchas veces desconocido o poco usado, se vincula directamente con los pseudo-elementos **:after** y **:before.**

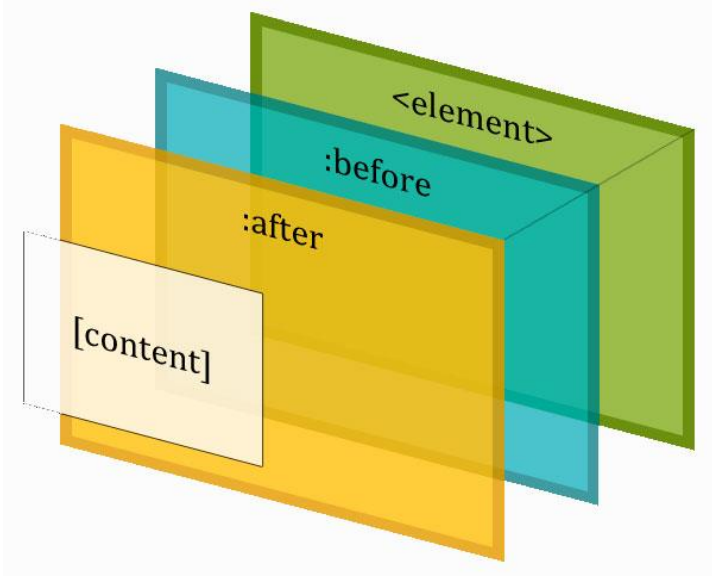


Fig. 10: el modelo de capas de un elemento, mostrando los pseudo-elementos :after y :before

Nicolas Gallagher explica en su web[52] con bastante detalle lo que voy a resumir en este apartado respecto a los muchos usos y posibilidades que

[52] Además, el autor del artículo original pone a disposición de la comunidad estos contenidos bajo licencia Creative Commons (Attribution-NonCommercial-NoDerivs 3.0 Unported), http://nicolasgallagher.com/multiple-backgrounds-and-borders-with-css2/

esto ofrece, y que básicamente consisten en la capacidad de contar con 3 lienzos (o superficies de trazado) por cada elemento del DOM, contando, además, con un excelente soporte por parte de los navegadores[53].

Funcionamiento

En esencia, creamos un pseudo-elemento cuando usamos la sintaxis **:after** o **:before** para definir un conjunto de reglas sobre un elemento y tratamos el resultado de estas definiciones como lo haríamos con cualquier otro elemento del DOM anidado dentro del elemento de destino.

Las capas generadas por estos pseudo-elementos son ubicadas detrás de la capa de contenido y pueden manejarse en cuanto a su posición mediante posicionamiento absoluto. Además, poseen una propiedad ***content*** donde podemos especificar el contenido a situar en la capa.

Esa propiedad, puede tener valores bien diversos: texto plano, códigos de escape expresados en Unicode u otro mecanismo de codificación y también gráficos.

La lista de posibles valores, es la siguiente:

- ***normal* | *none* (ninguno)**
- **<*string*> - Cadena**
- **<*url*> - Ubicación de un recurso** El valor asociado a una **url** se suele utilizar para referenciar un gráfico.
- **<*counter*> contador** Permite simular una numeración correlativa
- ***open-quote* – comillas (apertura)** Inclusión (o exclusión) de comillas
- ***close-quote* – comillas (cierre)** Lo mismo que el anterior

[53] Firefox 3.5+, Safari 4+, Chrome 4+, Opera 10+, IE8+.

- ***no-open-quote* -– comillas (sin apertura).** Ídem**.**
- ***attr* – Valor de un atributo**
- ***inherit* – heredado**

En el caso del valor de ***attr(<atributo>)*** resulta muy útil en ciertos contextos, como cuando –por ejemplo- definimos un estilo para salida por impresora y deseamos que las direcciones de internet se añadan al nombre del enlace (ya que el documento impreso no nos conduciría a ningún lado…)

Un ejemplo de utilización del valor de atributo podría ser el siguiente. Imaginemos que tenemos el siguiente código HTML:

```
<p>Etiam ornare magna et
   <a href="http://latinum.com">nisl convalps</a>
, sed apquam tortor vulputate</p>

Podríamos diseñar la siguiente regla CSS:

a:after {
content: ' ('attr(href)')';
}
```

Con lo que la salida correspondiente sería como la de la figura siguiente:

Fig. 11. Salida del código anterior mostrando la dirección añadida a continuación formateada entre paréntesis

Estos valores son los que con más frecuencia se utilizan al trabajar con estas capas "extra" que ofrece CSS. Pero, más allá de esto, disponemos de la posibilidad de incluir fondos y bordes. Y en esta última versión de CSS, se ha añadido la capacidad de establecer la ubicación y tamaño de esos elementos con más opciones de lo que hemos tenido nunca.

Para conseguirlo, estas capas se sitúan detrás de la capa de contenido, fijándolos a los puntos deseados mediante posicionamiento absoluto HTML. Los pseudo-elementos realmente no "contienen" nada en sí mismos, por lo que pueden ser estirados para situarse sobre cualquier área del elemento padre sin afectar al contenido de éste. Esto puede hacerse utilizando cualquier combinación de valores para los atributos ***top***, ***right, bottom, left, width,*** y ***height*** y eso es la clave de su flexibilidad.

Efectos disponibles

Utilizando solo un elemento se pueden crear efectos de paralaje, disponer de varios colores de fondo, varias imágenes de fondo normales o recortadas, imágenes de reemplazo, cajas expandibles usando imágenes para los bordes, columnas "falsas" de tipo fluido, imágenes existentes que desbordan la zona de contenido de la caja, bordes múltiples y otros efectos populares que generalmente requieren imágenes y/o el uso de presentación basada en HTML. También es posible incluir dos imágenes extra como contenido generado por cada capa.

La mayoría de elementos estructurales contiene elementos secundarios. Por lo tanto, a menudo podemos ganar otros 2 pseudo-elementos a utilizar en la presentación mediante la generación del primer elemento descendiente del elemento primario (y hasta de último). Además, se pueden utilizar cambios de estilo sobre el estado ***hover*** para producir efectos complejos de interacción.

Un primer ejemplo: Fondos múltiples y "*sprites*"

Partimos de un elemento contenedor que usará un posicionamiento relativo y contendrá su propio fondo y "*padding*". Gracias al posicionamiento relativo, el elemento actúa como marco de referencia cuando se posicionan los pseudo-elementos usando posicionamiento absoluto. Finalmente, un valor positivo de la propiedad ***z-index*** permitirá su ubicación correcta de cara al espectador (eje Z).

Además, vamos a utilizar una técnica antigua (de versiones anteriores), basada en "*sprites*", que supone utilizar un solo gráfico compuesto de múltiples partes para mostrar fragmentos de él en cada esquina.

Nuestro gráfico original es el siguiente:

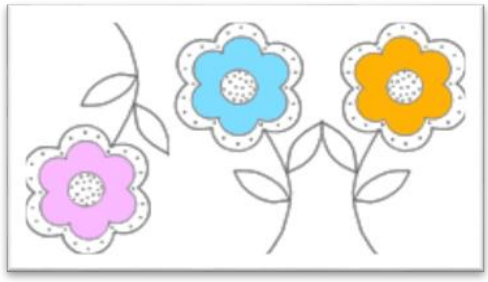


Fig. 12: Gráfico para "sprites"

Lo vamos a usar de manera que cada una de las flores aparezca en una esquina de nuestro elemento.

La parte HTML no puede ser más sencilla, ya que solo necesitamos un elemento contenedor (con un nombre asociado: *DivConSprites*), y un elemento interno que contiene un texto, cuyo espacio se irá amoldando a su contenedor, respetando los parámetros de distancias al borde, y haciendo que su altura crezca al decrecer su anchura.

```
<div id="DivConSprite">
    <p>
      Esto es el texto interior del elemento, contenido
      en un elemento -p-. Las reglas CSS que se le aplican
      (de la versión CSS 2.1), utilizan la técnica de
      sprites para añadir gráficos a 3 de las 4 esquinas
      del rectángulo contenedor.</p>
</div>
```

Y a continuación, definimos por separado los mecanismos para actuar sobre el elemento. Primero, su configuración de base:

```
#DivConSprite {
  position: relative;
  z-index: 1;
  width: 430px;
  padding: 85px;
  border: 1px solid #ddd;
  margin: 0 auto 40px;
  overflow: hidden;
  background: #fff url(Graficos/flowers.png) no-repeat -
150px 0;
}
```

Obsérvese que el fondo predeterminado es el gráfico, pero con posicionamiento negativo (el ancho de cada flor en la imagen es de 75px), de forma que nos quedamos con la tercera desde la izquierda, como imagen inicial. Esto nos genera una salida como la de la figura siguiente:

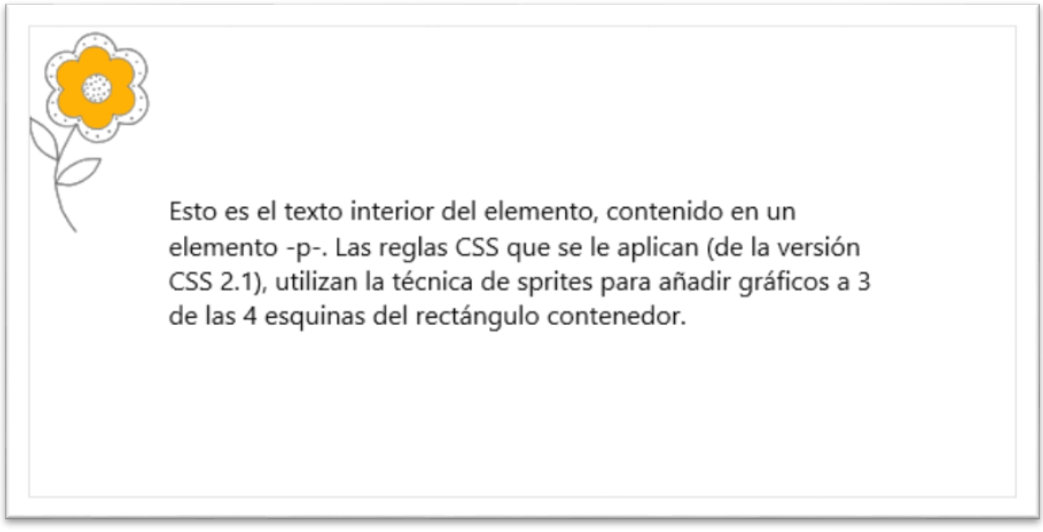


Fig. 13: Salida del código anterior en cualquier navegador actualizado

Si añadimos ahora código en común para los pseudo-elementos :***before*** y :***after***, podemos añadir otra flor a la derecha utilizando una técnica similar para estas capas añadidas:

```
#DivConSprite:before, #DivConSprite:after {
    content: "";
    position: absolute;
    z-index: -1;
    top: 0px;
    right: 0;
    width: 75px;
    height: 128px;
    background: #fff url(Graficos/flowers.png) no-repeat
-75px 0;
}
```

Lo que nos mostrará la flor situada en el centro, en la esquina de la derecha. El truco aquí es que la anchura de ***after*** y ***before*** se asigna exactamente a la anchura de cada uno de los *sprites*. A partir de ahí, el desplazamiento (-75px), hace que nos quedemos con la imagen del centro.

Finalmente, para completar el juego, podemos ubicar la primera de la izquierda en el gráfico para que ocupe la esquina inferior izquierda, y, además, tenga un grado de opacidad atenuado (volveremos más adelante sobre algunas de las reglas que estamos aplicando aquí):

```
#DivConSprite:after {
    top: auto;
    bottom: 0;
    right: 0;
    background-position: 0 0;
    opacity: 0.5;
}
```

La clave es que el estilo del pseudo-elemento :***after*** se define partiendo de la esquina opuesta (inferior derecha), con lo que el gráfico se mide desde el comienzo, y se posiciona en la esquina inferior derecha.

Con esto, tendríamos una salida final como la de la figura siguiente:

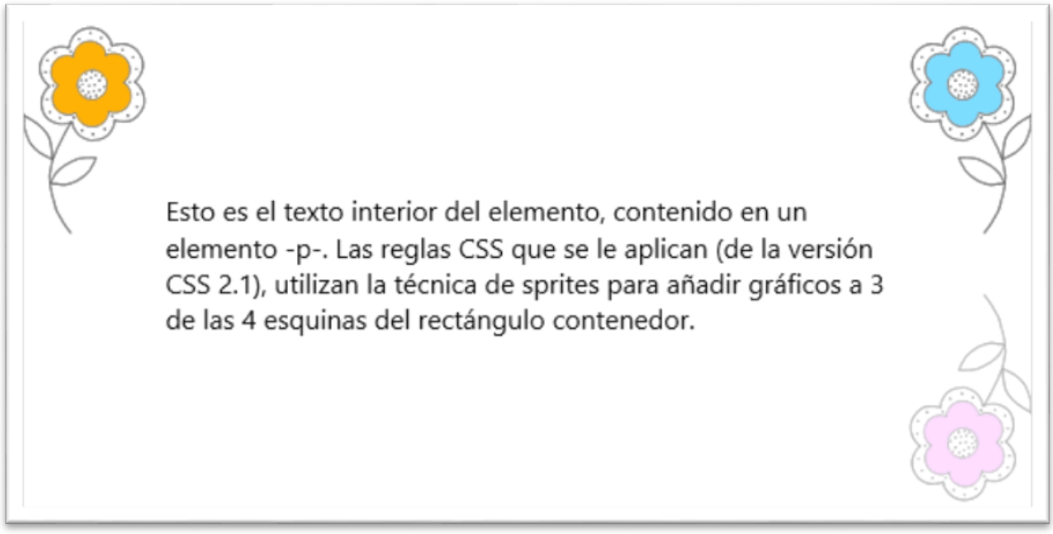


Fig. 14: la Salida del código anterior

Lo bueno de esta técnica es que funciona en todos los navegadores incluyendo IE8+, lo que nos permite utilizarla sin necesidad de *fallbacks*. La otra ventaja de utilizar *sprites*, es que estamos haciendo una sola llamada al servidor para descargar el gráfico y no 3, reduciendo el consumo del ancho de banda y el tiempo de respuesta del sitio.

También podemos conseguir otros efectos espectaculares, como hacer que la imagen de un de los pseudo elementos traspase los límites de la caja que lo contiene, jugando con los valores de la distancia al origen.

De todas formas, antes de comentar otros aspectos, es conveniente que revisemos el concepto de Modelo de Caja de un elemento y las propiedades que lo afectan, para luego proseguir con otras reglas que modifican este comportamiento.

El Modelo de Caja

Otro aspecto importante que comentar antes de continuar con CSS es lo que se denomina el Modelo de Caja de un elemento. Se denomina así al conjunto de reglas que el intérprete de CSS aplica a la hora de asignar a un elemento el espacio rectangular que le corresponde según las reglas que se apliquen a él.

En este modelo, cada elemento define un espacio de presentación y nos referimos a ellos mediante las palabras reservadas ***top, right, bottom*** y ***left.*** A su vez, cada uno de estos rectángulos está representado en el estándar CSS con las palabras **content, padding, border** *y* **margin**. El gráfico siguiente muestra el esquema de este modelo según lo presenta el documento oficial de la W3C disponibles en el sitio oficial[54]).

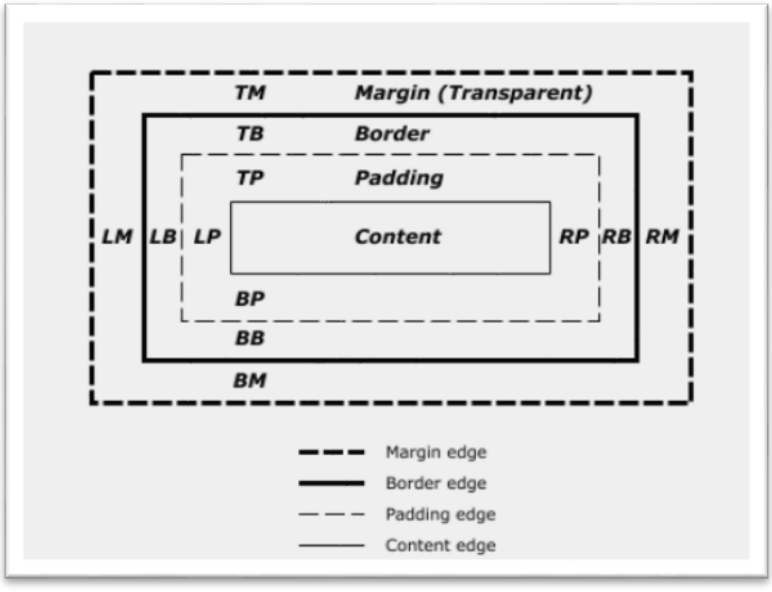


Fig. 15: Modelo de Caja de un elemento

En resumen, las propiedades (*width/height*) asignadas a un elemento

[54] http://www.w3.org/TR/CSS2/box.html

definen la anchura y altura de la caja ***content***. A su vez, ***padding*** establece la distancia hasta el borde y este borde puede tener un grosor. Además, y –contando desde el exterior del borde- la propiedad ***margin*** define el espacio requerido hasta cualquier elemento vecino y es transparente si no se indica lo contrario.

Con el nuevo modelo propuesto por CSS3, ahora podemos crear bordes personalizados, establecer colores, mecanismos de redondeo para las esquinas, y muchas cosas más.

La propiedad *display* y el modelo de caja

Antes de continuar con las reglas de CSS, es necesario aclarar el comportamiento de la propiedad ***display*** por la forma en que afecta al modelo de caja de los elementos. ***Display*** modifica la caja generada por un elemento, pudiendo tomar los siguientes valores (ver tabla adjunta).

Valores permitidos	Uno sólo de los siguientes: • **none** • **inline (predeterminado)** • **block** • **inline-block** • **list-item** • **run-in** • **table** • **inline-table** • **table-row-group** • **table-header-group** • **table-footer-group**

	• **table-row** • **table-column-group** • **table-column** • **table-cell** • **table-caption** **inherit**

Tabla 5: Valores para la propiedad display

El funcionamiento concreto es bastante complejo, y debido a ello resulta de los menos utilizados en la práctica, en parte, debido a que su implementación completa no es correcta en casi ningún navegador (aparte de que han aparecido nuevos posibles valores asociados a ***display*** en la especificación CSS3.

Aun así, su uso correcto resolvería no pocos problemas de distribución de los elementos en una página. Veamos brevemente la funcionalidad de cada posible valor.

none: No genera ninguna caja. El elemento no se muestra ni ocupa espacio alguno.

block: Genera una caja de estilo "bloque". Tal disposición hace que el elemento ocupe todo el espacio disponible (en anchura), independientemente de lo que ocupe su contenido. Los elementos con este tipo de *display* pueden tener asociados valores de anchura y altura.

inline: La caja generada es del estilo "en línea". El espacio ocupado es tan ancho como lo sea su contenido. Si, a un elemento generado mediante *display* se le asignan valores de anchura y altura, estos se ignoran en el proceso de "*rendering*" de la página.

inline-block: Tiene un comportamiento mixto. Se ubica en línea con el flujo natural de los elementos que le rodean, pero puede tener asignados valores tanto para la anchura como para la altura, lo que puede resultar

en una disposición como la de la figura siguiente:

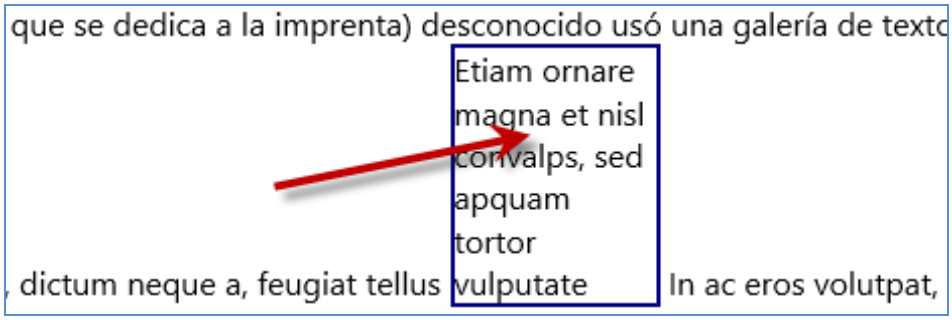


Fig. 16: elemento inline-block con un valor de anchura fijo (y altura automática)

list-item: fuerza a elemento a mostrarse como se tratase de un elemento de lista (***li***). Debido a esto, pueden utilizarse con ellos las definiciones propias de las listas, como *list-style*, *list-style-type*, *list-style-image* y *list-style-position*.

run-in: Mal soportado en todos los navegadores excepto Opera. Debería tener un comportamiento variable según el contexto de los elementos adyacentes.

table-*: El resto de propuestas se relaciona directamente con la disposición habitual a que estábamos acostumbrados al utilizar tablas, pudiendo definirse filas, columnas, celdas, grupos, cabeceras y títulos. Al igual que sucede con el anterior, el soporte es pobre en algunos navegadores, aunque puede usarse sin problemas en IE8+, Chrome y Opera.

Los nombres son suficientemente explicativos de cada función. (Nota: la regla ***empty-cells*** puede usarse para establecer el funcionamiento de celdas vacías, pudiendo asignarse los valores: ***hide*** (no mostrar fondo ni bordes), *show* (mostrar normalmente. Valor predeterminado) e ***inherit*** (el valor se hereda del elemento ***parent***).

inherit: Se hereda el valor del elemento contenedor (***parent***).

El nuevo modelo de cajas propuesto por CSS 3

Hacemos un inciso aquí para anticipar otro de los cambios de esta

versión, ya que tiene que ver con el tamaño de las cajas asignadas visualmente a cada elemento por el DOM: la nueva propiedad ***box-sizing*** permite especificar que, cuando un elemento disponga de una definición para su anchura (***width***), los posibles valores de las propiedades ***border*** y ***padding*** se incluyan como parte de la anchura (esto no afecta al valor de la propiedad ***margin***).

Por poner un ejemplo concreto, si asignamos a un elemento una anchura de 200px, 5px para el borde (***border-width***) y 15px para el ***padding***, la anchura total del elemento seguirá siendo de 200px, y el espacio disponible para el contenido será de 180px, desde el momento que utilicemos la propiedad ***box-sizing*** con un valor ***border-box***.

El siguiente código fuente muestra este caso: asigna estos valores a dos párrafos con borde, de manera que podamos observar la diferencia:

```
p:nth-of-type(4) {
    font-family: Fuente4;
    width: 400px;
    border-width: 5px;
    border-style: solid;
    border-color: navy;
    padding: 15px;
}

p:nth-of-type(5) {
    font-family: Fuente4;
    width: 400px;
    border-width: 5px;
    border-color: navy;
    border-style: solid;
    padding: 15px;
    box-sizing: border-box; /* Añadimos border-box */
}
```

En el ejemplo, los valores son idénticos para ambos párrafos, con la salvedad de que en el segundo, establecemos la propiedad ***box-sizing*** a ***border-box*** (los otros valores posibles son ***padding-box*** y ***content-box***). Esto produce una diferencia visible en la salida, como vemos en la figura

17:

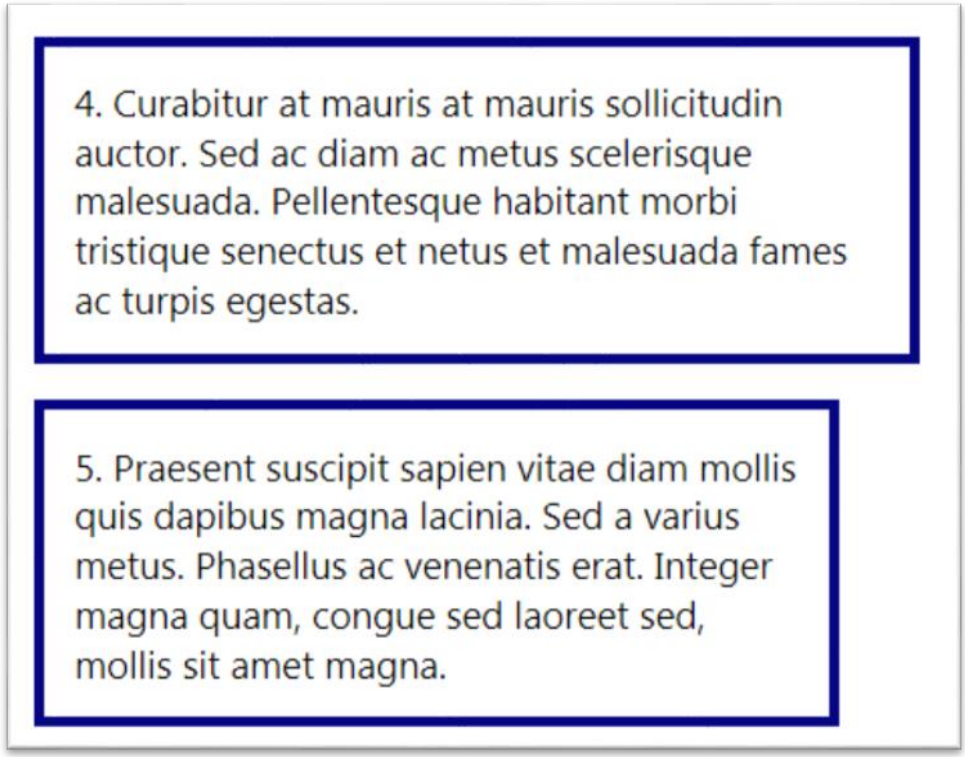


Fig. 17: Diferencia en los tamaños de las cajas con la propiedad box-sizing vistos en IE10/11

Y esa no es la única modificación. Otro cambio interesante se ha producido en el tratamiento del texto respecto a la propiedad de desbordamiento. A las propiedades existentes de versiones anteriores, se han añadido ***overflow-x*** y ***overflow-y***, que permiten indicar el comportamiento del texto desbordado tanto en horizontal como en vertical.

El comportamiento es simple: sin un texto no cabe en horizontal y especificamos la propiedad ***overflow-x: scroll;*** aparecerá una barra de desplazamiento horizontal en la parte inferior. Y lo mismo podemos hacer para la misma situación en vertical.

Las nuevas definiciones en CSS 3

Una vez hecha esta revisión de los fundamentos de CSS (de versiones anteriores, en la mayor parte de los casos), vamos a centrarnos en las novedades de esta versión, que son muchas, y en muchas áreas distintas (de ahí la división en módulos del estándar).

Por orden alfabético, cabe mencionar: *Borders, Backgrounds, Box, Color, Content, Media, Opacity, Text, Transformations, Transitions* y *Web Fonts*. Vamos a analizar las más significativas entre ellas, y ver unos ejemplos de utilización.

Modificaciones estructurales

Uno de los aspectos a considerar es que algunas modificaciones introducidas por CSS3 son de tipo global o estructural, y por lo tanto, afectan a muchas otras. Ese es el caso del atributo ***opacity*** (que admite un valor decimal entre 0 y 1), uno de los más esperados, que funciona en cualquier elemento, y no solamente vinculado a la regla ***background*** o a las imágenes. Otro tiene que ver con la forma de expresar las unidades de color, como vamos a ver a continuación.

Colores

En el apartado de colores, podemos seguir haciendo las selecciones exactamente igual que hasta ahora, y siguen existiendo un conjunto de colores predeterminados ya definidos por los estándares HTML 4 y SVG. La novedad es que ahora podemos utilizar el modelo **HSL**[55] (Matiz, Saturación, Luminosidad), para definir un color en términos de sus componentes constituyentes.

La lista completa de opciones para el parámetro Color, es la que muestra la Tabla 6, en la que se han añadido algunos elementos nuevos para facilitar las definiciones. En concreto, son nuevos en CSS 3: RGBA, HSL,

[55] http://es.wikipedia.org/wiki/Modelo_de_color_HSL

HSLA.

Nombre	Expresión	Ejemplo
Color con nombre	**<nombre del color>**	**navy**
Transparent	***transparent***	**–**
Current color	**currentcolor**	**–**
RGB-hexadecimal	**#rrggbb, #rgb**	**#ff7f50, #f90**
RGB-decimal	**rgb(rrr,ggg,bbb)**	**255,127,80**
RGB-porcentaje	**rgb(rrr%,ggg%,bbb%)**	**rgb(100%,50%,31%)**
HSL	**hsl(hhh,sss%,lll%)**	**hsl(16,65%,100%)**
RGBA	**rgba(rrr,ggg,bbb,d.d)**	**rgba(255,127,80,.86)**
HSLA	**hsl(hhh,sss%,lll%,d.d)**	**hsl(16,65%,100%,.23)**

Tabla 6: Valores posibles de los colores y formas de expresión

Como vemos, existe ahora la posibilidad de establecer colores en función del color activo en la propiedad ***color*** correspondiente al selector. Por ejemplo, si el color actual es ***Navy***, podemos hacer que el color del borde sea igual, seleccionando:

```
color:navy;
border-color: currentcolor;
```

Más utilizada es la opción de indicar el color mediante sus colores básicos RGB (componente tricolor) o RGBA (componente tricolor más canal Alfa), donde, como se ve en el cuadro, podemos indicar los valores hexadecimales, los decimales, o valores porcentuales.

Novedades que afectan a la Caja de los Elementos

Muchas de las novedades de esta versión afectan a la forma en que podemos mostrar ahora la caja de un elemento, y por tanto es aplicable todo lo visto hasta este momento. Algunas de las más populares se refieren a los fondos, los bordes, las sombras.

Fondos (Propiedad *Background*)

Por supuesto, la propiedad ***background***, puede adoptar cualquiera de los colores creados mediante las definiciones de la tabla anterior.

En las interfaces de usuario actuales, es muy común disponer de superficies cuyo fondo lo forma un gradiente de color, utilizan texturas basadas en imágenes, repiten patrones, y muchos otros efectos por el estilo.

El problema reside aquí en que la implementación es, por el momento, ligeramente diferente dependiendo del navegador por lo que tendremos que utilizar las extensiones que vimos al principio del capítulo. Y, en algunos casos, no existe un equivalente propio en el lenguaje CSS, por lo que hay que recurrir al elemento **<*canvas*>** o a **SVG** para conseguir el efecto deseado.

Aunque muchas propiedades relacionadas con ***background*** han sido ya implementadas en los navegadores, existen algunas que dependen del navegador, como muestran las imágenes siguientes al utilizarlo desde el editor de código CSS de Visual Studio 2012:

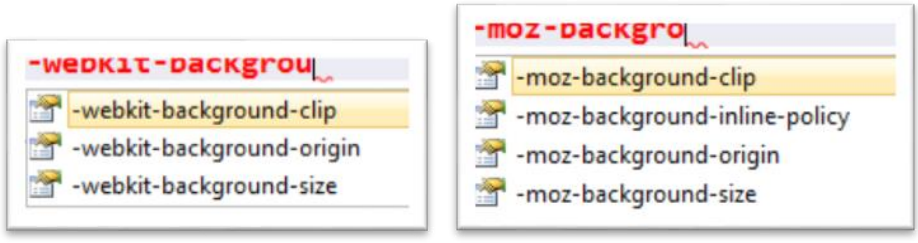


Fig. 18: Implementaciones de la misma propiedad en la extensión –webkit (Chrome/Safari) y Mozilla (FireFox).

Las buenas noticias son que, con la variedad de técnicas disponibles en este momento, muchas veces, nos encontraremos con varias formas de hacer lo mismo.

Por ejemplo, si queremos implementar un fondo de degradado de manera que se vea en todos los navegadores implicados, necesitaremos una definición como la siguiente:

```
footer {
    /* W3C Markup, IE10 Preview */
    background-image: linear-gradient(to right bottom,
rgb(255, 255, 255) 0%, rgb(0, 163, 239) 100%);
    /* IE10 Release Preview */
    background-image: -ms-linear-gradient(Gray 0%, white
100%);
    /* Mozilla Firefox */
    background-image: -moz-linear-gradient(top left,
#FFFFFF 0%, #00A3EF 100%);
    /* Opera */
    background-image: -o-linear-gradient(top left,
#FFFFFF 0%, #00A3EF 100%);
    /* Webkit (Safari/Chrome 10) */
    background-image: -webkit-gradient(linear, left top,
right bottom, color-stop(0, #FFFFFF), color-stop(1,
#00A3EF));
    /* Webkit (Chrome 11+) */
    background-image: -webkit-linear-gradient(top left,
#FFFFFF 0%, #00A3EF 100%);
}
```

Donde, como vemos, se utilizan extensiones personalizadas para obtener el mismo resultado, a excepción de IE10 que ya cumple con la especificación del estándar. La salida será similar a la que ofrece la Figura 19, para todos los navegadores que estudiamos.

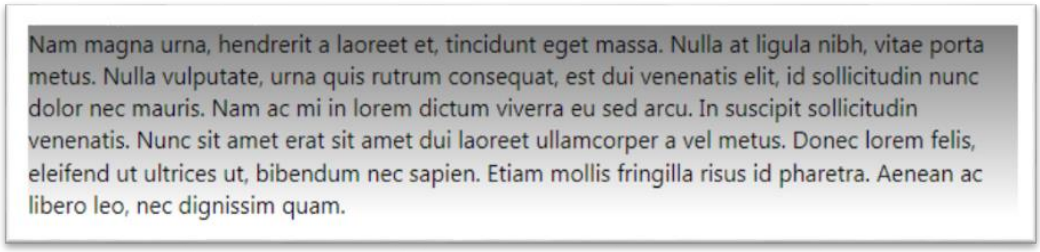


Fig. 19: Salida del código anterior aplicado a un párrafo, mostrando el nivel de gradiente

De hecho, podemos conseguir efectos similares a los que hemos visto para el elemento ***canvas*** y SVG, utilizando solamente CSS 3. Por ejemplo, el siguiente código fuente, produce en IE10 una variante radial (en vez de lineal) del efecto de degradado, e incluye varios puntos de control (***colorStops***):

```
    background-image: -ms-radial-gradient(50% 50%,
circle, maroon 9%, yellow 50%, green 100%);
```

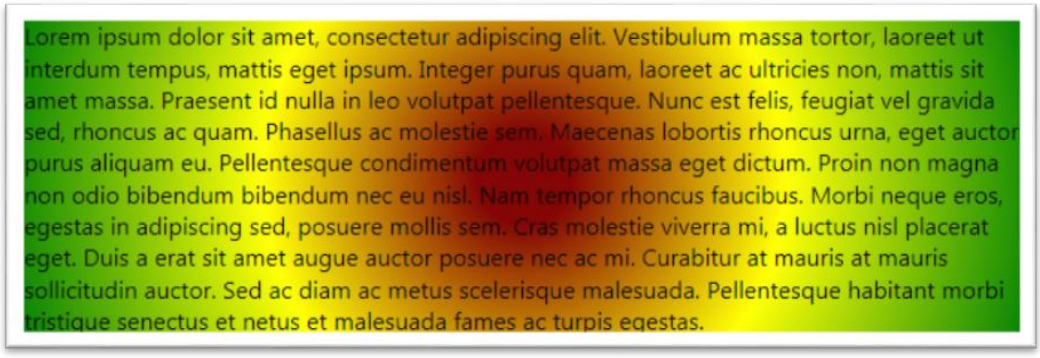


Fig. 20: Efecto de gradiente radial en IE10

Las propiedades nuevas en CSS3 relacionadas con la definición de fondos y la propiedad ***background*** son las siguientes:

- **background-clip (nueva en CSS3: recorte)**
- **background_origin (nueva en CSS3: desplazamiento desde el origen)**

- **background-size (nueva en CSS3: tamaño del fondo)**
- **Múltiples backgrounds (nuevo en CSS3)**

Una vez más, Visual Studio 2012 dispone de soporte para programación de estas características, junto a sus posibles valores y significados, como podemos ver en la figura 21:

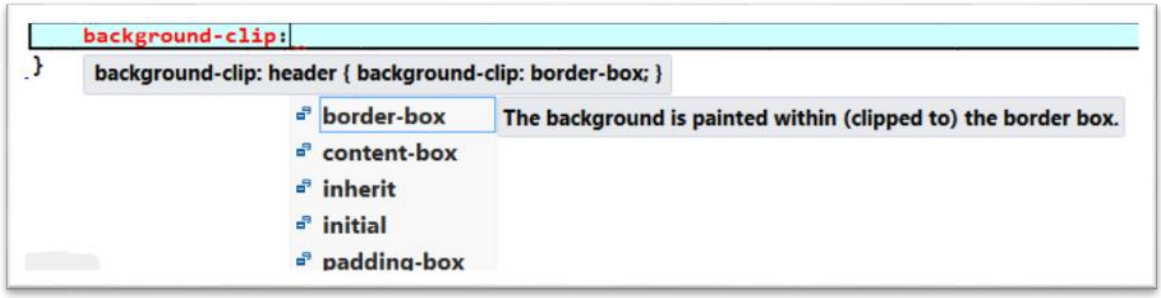


Fig. 21: El editor de CSS 3 en V. Studio 2012 mostrando las propiedades de background-clip

background-clip

Sirve para establecer si el fondo se extiende hasta incluir los bordes o no. El valor predeterminado es ***border-box***, que significa que sí se extiende, pero podemos utilizar igualmente las palabras reservadas ***padding-box*** y ***content-box*** (además de las habituales ***initial*** y ***inherit***), para limitarlo de forma similar a como hemos visto en el caso anterior del fondo.

background_origin

Relacionada con la anterior, disponemos ahora de la propiedad ***background-origin*** que nos permite indicar a partir de dónde se empieza a calcular la posición. Los valores posibles son: ***border-box, padding-box*** y ***content-box***, que hacen referencia respectivamente, al borde del contenedor, al borde descontada la cantidad de ***padding*** (si lo hubiera), y al borde delimitado por el propio contenido.

background-size

Mediante **background-size** podemos asignar un conjunto diverso de valores para el tamaño de lo que hayamos establecido como fondo del elemento: ***auto*** (se calcula en función del contexto), ***inherit*** (heredada de su contenedor), ***initial*** (valor predeterminado), una expresión de ancho/alto (para un tamaño establecido), o un porcentaje sobre el original. Además, también podemos utilizar las constantes ***cover*** y/o ***contain***:

- En el primer caso, la imagen de fondo ocupa todo el espacio disponible (si es más grande que el fondo), pudiéndose perder parte de la imagen.
- En el segundo, la imagen se ajusta al contenido, pero, respetando la relación de aspecto, por lo que podrían quedar partes del fondo sin cubrir por la imagen (todo esto, suponiendo que hemos optado por el parámetro ***no-repeat***, para impedir que se use la configuración predeterminada).

Por ejemplo, supongamos que queremos que exista una imagen de fondo dentro de uno de los elementos *** *** que contienen texto. Podríamos definir las siguientes reglas CSS para ese elemento:

```
div
{
    background:url(../Imagenes/kfilereplace.png) no-
repeat;
    background-size:contain;
    background-clip:border-box;
    background-origin:border-box;
}
```

Esto provocaría la aparición de la imagen en segundo plano, pero, además, un efecto dinámico: según el usuario modifique el tamaño del navegador, así lo hará igualmente el tamaño del texto que contiene el *** ***, y por la propiedad ***background-size***, el tamaño de la imagen, de

forma que siempre se mantendrá a una distancia fija de los bordes, y sufrirá modificaciones de escala para adaptarse al contenedor.

La figura 22 muestra el funcionamiento en ejecución en IE10/11, con dos anchuras distintas del navegador:

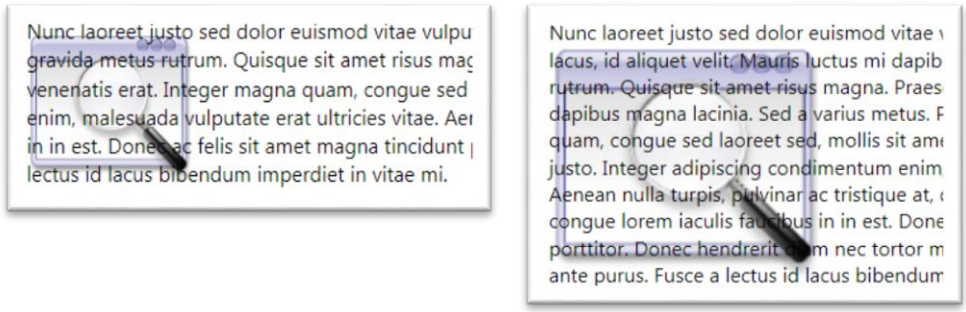


Fig. 22: Dos capturas de la salida en IE10/11 del código anterior.

Finalmente, es posible indicar más de una imagen con la característica que hemos llamado "Múltiples ***backgrounds***". Para ello, basta con indicar tantas imágenes de fondo como deseemos, pero separadas por comas. Si queremos establecer ubicaciones concretas para las imágenes, solo tenemos que indicárselo en el mismo orden (y también separadas por comas) en la propiedad ***background-position.***

El código siguiente cumple con ese objetivo básico, cambiando la ubicación del primer gráfico a la derecha del elemento, y dejando el segundo en la ubicación predeterminada (izquierda):

```
div
{
 background:url(../Imagenes/kfilereplace.png) no-repeat right,
            url(../Imagenes/DownloadFile.png) no-repeat;
 background-size:contain;
}
```

Lo que genera la salida que podemos ver en la figura 23:

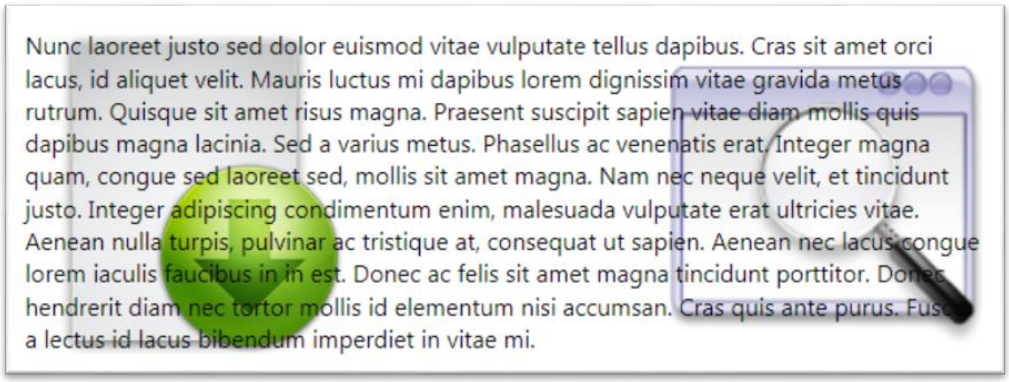


Fig. 23: Salida del código anterior en IE10 mostrando las dos figuras de referencia.

Bordes

Con el nuevo modelo propuesto por CSS3, ahora podemos crear bordes personalizados, establecer colores, mecanismos de redondeo para las esquinas, y muchas cosas más.

Comenzamos con las típicas esquinas redondeadas. Para conseguir este efecto, indicaremos los valores del radio de la circunferencia para cada una de las aristas del rectángulo, o bien un valor idéntico para todos ellos.

La siguiente combinación de selectores, hará que el segundo párrafo de la página que lo utilice aparezca con un borde rojo de 3 píxeles, redondeado solamente por la izquierda (Se han añadido las extensiones de los navegadores **–webkit** y **–moz** para que la salida sea idéntica en todos ellos):

```
p:nth-of-type(2) {
    padding: 5px;
    border: 3px solid red;
    -webkit-border-top-left-radius: 10px;
    -webkit-border-bottom-left-radius: 10px;
    -moz-border-radius: 10px 0 0 10px;
    border-radius: 10px 0 0 10px;
}
```

Lo que nos muestra una salida como la de la figura 24.

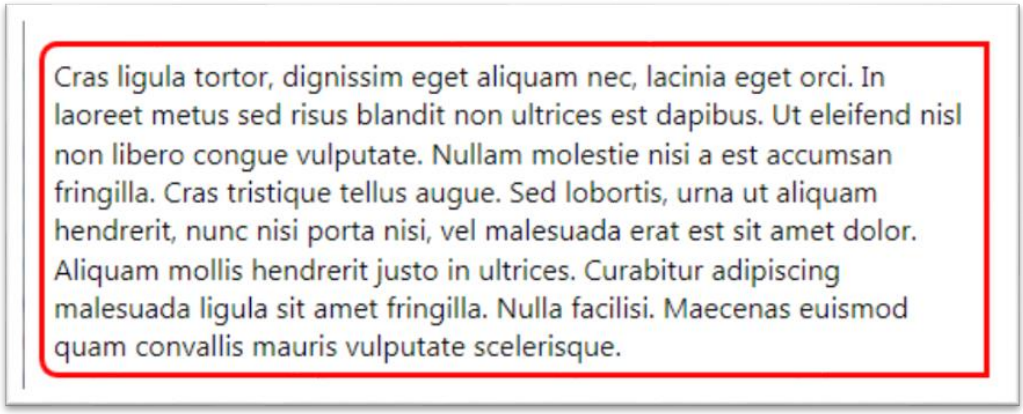


Fig. 24: Definición de un borde para el segundo párrafo de una página.

También se pueden conseguir formas elípticas, en lugar de redondeadas sustituyendo el único valor del radio, por dos valores (uno para cada foco de la elipse).

Al aplicar estas nuevas opciones, deberemos de tener en cuenta la ubicación de la caja y los aspectos antes comentados, ya que el borde podría interferir en el contenido (de un elemento de texto, por ejemplo). Tal es el caso del siguiente ejemplo, donde diseñamos un borde irregular, que utiliza diferentes valores para el radio y un modelo ***dotted*** (de puntos), a la salida final:

```
p:nth-of-type(1) {
    border-color: brown;
    border-radius: 80px 35px 80px 35px;
    border-style: dotted;
    border-width: 16px;
}
```

Lo que genera la salida que podemos ver en la figura 25:

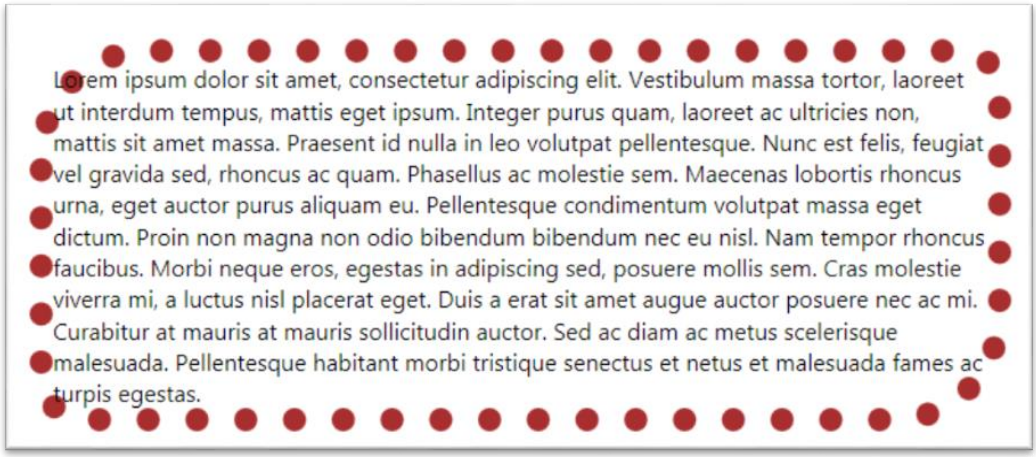


Fig. 25: Salida de bordes redondeados no uniforme, del código anterior

Bordes definidos mediante imágenes

Otra posibilidad vinculada con los bordes consiste en utilizar la idea de textura (una imagen de relleno), utilizando la propiedad ***border-image*** o sus equivalentes ***–moz-border-image*** y ***-webkit-border-image.*** En realidad, el estándar establece un conjunto posible de valores para poder refinar la salida, de forma que la propiedad se puede desglosar en dos: ***border-image*** y ***border-corner-image***, que, a su vez, pueden especificarse de forma individual para cara arista, disponiendo de las siguientes propiedades en total[56]:

- **`border-image:`**
  - **`border-top-image`**
  - **`border-right-image`**
  - **`border-bottom-image`**
  - **`border-left-image`**

- **`border-corner-image:`**
  - **`border-top-left-image`**
  - **`border-top-right-image`**

[56] Para una explicación detallada del funcionamiento estándar de los bordes, visitar http://www.w3.org/TR/2002/WD-css3-border-20021107/#the-border-image-uri

- **`border-bottom-left-image`**
- **`border-bottom-right-image`**

De esta forma, podemos definir la salida de una definición ***border*** a partir de cualquier gráfico para que sea utilizado por el mecanismo de interpretación visual.

Naturalmente, la forma resultante dependerá de los parámetros y de la propia imagen, por lo que conviene experimentar primero y tener en cuenta las distintas implementaciones disponibles. En el ejemplo canónico indicado en el enlace de la página anterior, se utilizan 8 pequeños fragmentos de imagen, para obtener una salida como la que muestra la Figura 26:

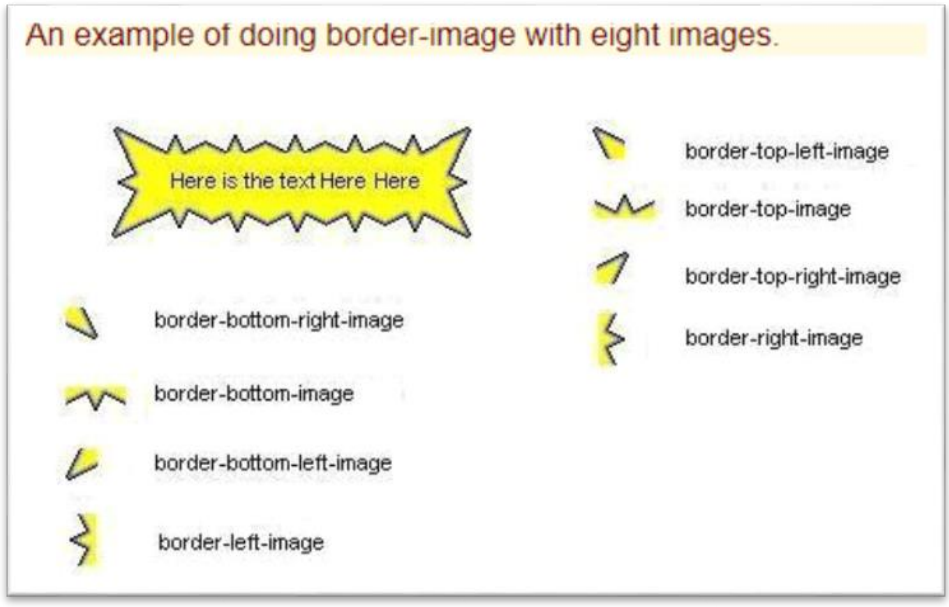


Fig. 27: Ejemplo oficial de la W3C ilustrando algunas posibilidades de la utilización de bordes con imagen.

Efectos de sombreado

Otra posibilidad interesante, consiste en la definición de sombras, pudiendo ser tanto externas (lo habitual) como internas (para ciertos efectos especiales).

La propiedad se llama ***box-shadow*** y admite los siguientes parámetros:

- **X, Y**: desplazamiento desde el rectángulo (los valores positivos desplazan la sombra abajo y a la derecha, mientras los valores negativos lo hacen arriba y a la izquierda).
- ***Blur***: grado de distorsión de la sombra (0 de forma predeterminada)
- ***Spread***: Valor de longitud que indica el punto en que empieza a desvanecerse la sombra (0 de forma predeterminada)
- ***Color*:** Cualquier definición de color válida de las estudiadas en este mismo capítulo (si no es establece, se asume ***transparent***).
- ***Inset***: Causa que la sombra se pinte hacia dentro del rectángulo (es opcional).

Podemos hacer una prueba sobre nuestra página cambiando la forma en que se muestra el primer párrafo, para que utilice alguna de estas opciones:

```
p:nth-of-type(1)
{
    border: 2px solid blue;
    box-shadow: 10px 10px 5px #888888;
    -webkit-box-shadow: 10px 10px 5px #888888; /*
Safari */
}
```

Donde, además del borde, estamos indicando una sombra que se traza con un desplazamiento de 10 píxeles desde la esquina inferior izquierda, un grado de distorsión de 5px y un color gris oscuro. La salida se aprecia en la figura 28.

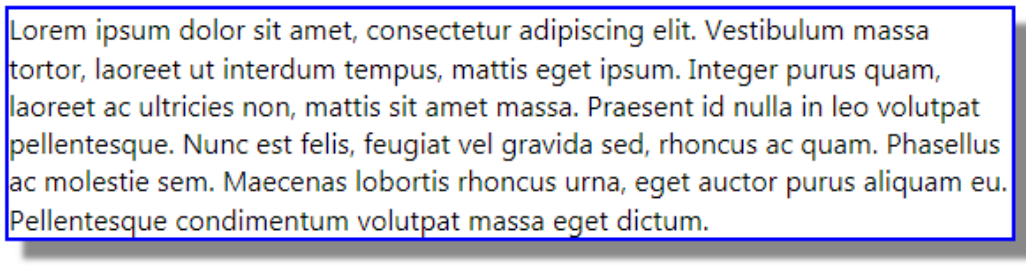


Fig. 28: Salida del código que define una sombra y un borde para el primer párrafo.

De la misma forma, podemos definir más de una sombra, expresando cada una de ellas separada por comas de la anterior.

De hecho, hay muchos más efectos relacionados con esta definición. Podemos optar por dos modos totalmente distintos: el modo estándar, donde cada configuración de sombra se define de manera individual, y el modo ***block***, que consigue el efecto final de una figura tridimensional, haciendo coincidir todos los parámetros de color.

Podemos ver la diferencia en el resultado mediante unos sencillos ejemplos. Supongamos que disponemos de un elemento vacío que queremos que adopte la forma de un gráfico con sombra, en el modelo estándar (básico).

El siguiente código fuente, consigue un efecto de redondeado en los bordes, un fondo gris para el texto, y una sombra de color verdoso, con unos valores de desplazamiento, difuminado de la sombra, etc.:

```
 p:nth-of-type(2) {
        padding:30px;
        background-color: rgb(204, 204, 204);
        border-radius: 58px;
        border-width: 0px;
        box-shadow: 10px 10px 6px 5px hsla(59, 72%, 25%,
0.84);
    }
```

La sombra tiene un color difuminado establecido por el canal Alfa (0.84), más unos valores de ***offset*** (desplazamiento vertical y horizontal), generando una salida como la que vemos en la figura 29 (IE10/11):

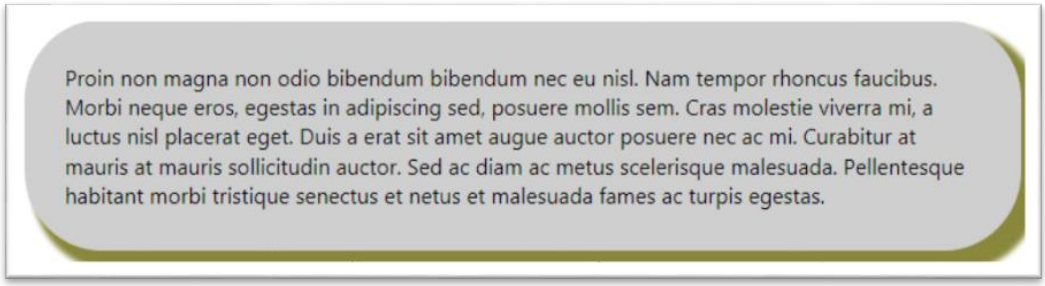


Fig. 29: Salida del código anterior en IE10/11

Si le añadimos una segunda sombra, y cambiamos algunos parámetros, de color, podemos fácilmente obtener una salida como la de la figura 30:

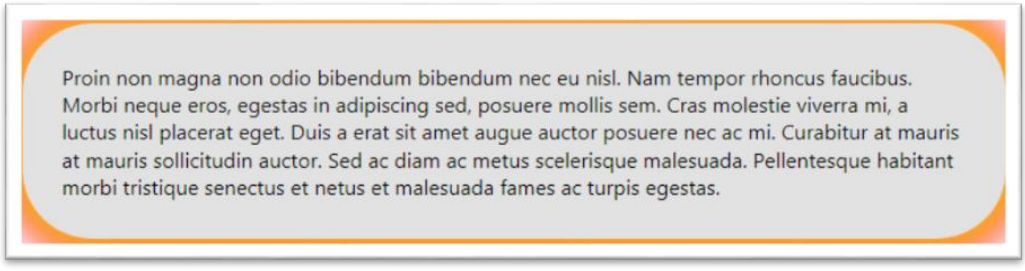


Fig. 30: Mismo elemento formateado con efecto "glow"

Que se corresponde con el siguiente código fuente:

```
p:nth-of-type(2) {
    padding:30px;
    background-color: rgb(224, 224, 224);
    border-radius: 53px;
    border-width: 12px;
    box-shadow: 0px 0px 27px 5px rgba(241,245,0,0.71),
0px 0px 50px 12px rgb(255,13,5);
    }
```

Y también podemos obtener el efecto de un bloque sólido,

tridimensional, optando por un código fuente similar a este:

```
p:nth-of-type(2) {
    padding:30px;
    background-color: rgb(176, 214, 133);
    border-radius: 48px;
    border-width: 0px;
    box-shadow: 1px 1px rgb(53,147,36),
                2px 2px rgb(53,147,36),
                3px 3px rgb(53,147,36),
                4px 4px rgb(53,147,36),
                5px 5px rgb(53,147,36),
                6px 6px rgb(53,147,36),
                7px 7px rgb(53,147,36);
}
```

Y el efecto de bloque, podemos verlo en la figura 31, basándonos en que el tono de la sombra se define varias veces y se desplaza línea por línea:

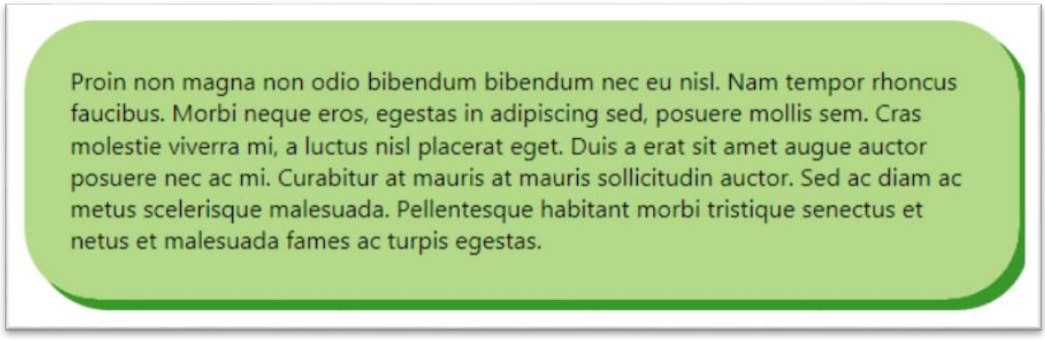


Fig. 31: Efecto de sombra "tipo bloque" o sólida, usando varias definiciones encadenadas

El soporte de navegadores es bastante completo en las versiones actuales.

Manipulación de Texto

Otra de las posibilidades que más reclamaban los desarrolladores era un mayor conjunto de opciones relativas al texto. Con la nueva especificación, disponemos de nuevos formatos de tipos de letra, efectos especiales, y diversos mecanismos de ajuste fino, *kerning,* etc.

Además, ya que los textos son inseparables de la tipografía, disponemos ahora de muchas posibilidades en este campo, incluyendo nuevos formatos estándar de tipos de letra.

Tipos de letra

Es un principio bien establecido por los diseñadores Web que una tipografía adecuada hace el documento más legible, más ajustado al mensaje que pretende transmitir y puede marcar el carácter corporativo que muchas empresas y profesionales buscan como parte de su mensaje en la red. Un texto claro invita a la lectura.

Ahora, podemos personalizar nuestros textos mediante fuentes descargables (gracias a la nueva opción ***Web Fonts***), de forma que tengamos la garantía de que el texto tiene exactamente (o casi…) la misma apariencia sin importar el dispositivo, al tiempo que conseguimos que la diferencia entre el diseño teórico y el resultado final, sea lo más ajustada posible.

Indicación del texto seleccionable

El nuevo selector extendido **-ms-user-select** es una propiedad que permite controlar dónde pueden los usuarios seleccionar texto en sus páginas web (o en aplicaciones estilo Metro que utilicen JavaScript).

Actualmente, la propiedad ***user-select*** no forma parte de ninguna de las especificaciones de CSS. Originalmente estaba propuesta en el módulo de interfaz de usuario para CSS3, pero ese módulo ha sido remplazado por el de interfaz de usuario básica para CSS3, que no contiene esta propiedad.

Sin embargo, otros exploradores admiten sus propias versiones prefijadas de esta propiedad.

Los valores posibles son (fuente MSDN):

none	Impide que la selección comience en ese elemento. No impide que una selección existente entre en el elemento.
element	Permite que la selección comience en el elemento, pero la misma debe estar dentro de los límites de dicho elemento.
text	Permite que la selección comience en el elemento y se extienda más allá de los límites de este.

Tabla 7: valores posibles de user-select

Fuentes y formatos

La posibilidad de descargar fuentes personalizadas nos abre un abanico interesante de opciones de presentación. Claro que, como la alegría no podía ser completa, nos encontramos con que distintos navegadores soportan formatos diferentes.

Los formatos más populares de uso en la Web hoy día son:

- ***WOFF*** (*Web Open Font Format*)[57] – Recomendado en CSS3
- ***TTF/OTF*** (*True Type Font/Open Type Font*[58])
- ***SVG*** (*Scalable Vector Graphics*) - ya comentado- puede contener fuentes
- ***EOT*** (*Embedded Open Type*)

El primero es el nuevo jugador en la partida, y se ha desarrollado específicamente para la Web, contando con el soporte de la mayoría de la industria. Se trata de tipografías TTF con metadatos, que se optimizan para su distribución por la red. Lo bueno de este formato es que contamos con un soporte prácticamente completo en todos los navegadores analizados.

[57] http://www.w3.org/Submission/WOFF/

[58] http://www.microsoft.com/typography/otspec/

El resto, aparte de no ser especificaciones oficiales de la W3C, tiene una aceptación desigual en los navegadores, por lo que la recomendación general es utilizar el primero cuando queramos usar fuentes propias. El último documento oficial disponible sobre fuentes es bastante reciente, y explica con detalle los avances actuales del estándar y las mejores recomendaciones de uso[59].

> **Nota**: Si el lector quiere utilizar sus propias fuentes sin tener problemas de licencias, podemos recomendar el sitio **Font Squirrel Web** (http://www.fontsquirrel.com/) que no solo dispone de más de 500 fuentes gratuitas, sino que habilita la herramienta **@font-face Kit Generator** (http://www.fontsquirrel.com/fontface/generator), que permite convertir cualquier tipo de fuente OTF o TTF en los formatos EOT, SVG y WOFF.

Para indicar que un fuente es descargable, usamos la regla (equivalente a una directiva) **@font-face**, que permite definir conjuntos de fuentes, indicando su nomenclatura, ubicación, tipo, etc.

En el caso de que queramos facilitar una fuente alternativa podemos establecer más de una definición de **@font-face**, sabiendo que los navegadores que no comprendan ese formato, simplemente ignorarán la orden.

La sintaxis formal es del tipo:

```
@font-face {
```

[59] http://www.w3.org/TR/css3-fonts/#font-resources

```
  font-family: MisFuentes; /* Identificador de familia de
fuentes */
  src: local(Fuente1), /* usa Fuente1 si está disponible
localmente */
  url(Fuente1)  format('woff') ; /* si no, la descarga */
 /* …y se indica el formato */
}
```

Lo que en una situación real, donde queremos utilizar más de una fuente, nos ofrece la posibilidad de expresarlo en la forma que vemos en el siguiente código:

```
@font-face {
font-family: "Fuente1";
src: url('../Fuentes/tt0815m_.woff');
}

@font-face {
font-family: "Fuente2";
src: url('../Fuentes/tt0727m_.woff');
}

p:first-of-type {
    font-size:16pt;
    font-family:Fuente1;
}

p:nth-of-type(2) {
    font-size:16pt;
    font-family:Fuente2;
}
```

Lo importante para el desarrollador es la ubicación correcta de la fuente (ya sea local o remota). La declaración ***font-family*** es simplemente un identificador personalizado que no tiene por qué coincidir con el nombre físico o lógico de la fuente.

De hecho, el código de más arriba aplicado a nuestro ejemplo de varios párrafos tipo "*Lorem Ipsum*", genera en IE10 (y el resto de navegadores),

la siguiente salida:

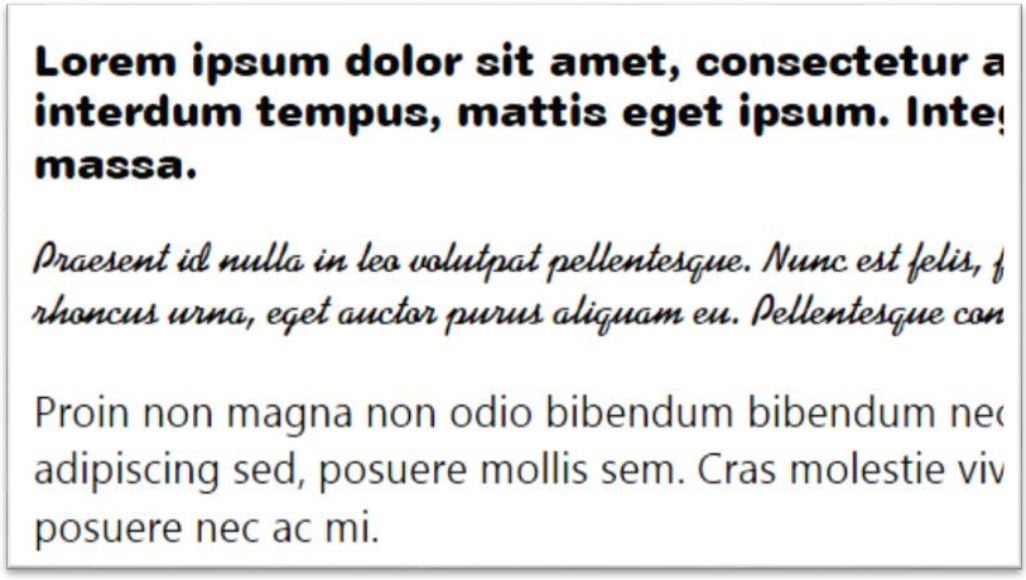


Fig. 32: Prueba de la directiva @font-face en IE10/11

Hasta aquí, es algo que teníamos disponible en las versiones anteriores de CSS (a excepción de la recomendación WOFF). Pero, uno de los problemas actuales con las fuentes reside en las dificultades de legibilidad, cuando tenemos una estructura de documento, y por las razones que sean, el navegador muestra el texto con una letra diferente de las programadas por nosotros.

Unidades de medida de las fuentes

También en este apartado ha habido novedades interesantes. Y no está de más recordar primero cuáles son las unidades de medida utilizadas para calcular el tamaño de una fuente. Empezando por distinguir las unidades absolutas de las relativas.

Las distintas unidades absolutas que podemos usar son las siguientes:

- **in**: **pulgadas** ("*inches*"). Equivalen a 2.54 centímetros.

- **cm**: **centímetros**.
- **mm**: **Milímetros**.
- **pt**: el **Punto** equivale a 0.014 pulgadas
- **pc**: **Picas**, equivalen a 12 puntos.

Y las unidades relativas, que permiten establecer con mayor coherencia el tratamiento de las fuentes de un sitio son estas:

- **px**: la más utilizada es el **Pixel**, la cantidad de puntos en los que se divide el monitor.
- **ex**: se refiere a la altura de la "x" del tipo de letra usada
- **em**: también llamada "cuadratín" es relativa al tamaño de letra definido por **font-size**.

Ésta última, es una de las más utilizadas, precisamente por la posibilidad de establecer el tamaño de una fuente dependiendo de otros valores en cascada (heredados de otros elementos).

La novedad es que ahora disponemos de una nueva unidad, ***rem***, que permite establecer el tamaño en función del definido para el elemento ***<html>***, que es la raíz del documento (de ahí el nombre, una abreviatura de ***root*** *y* ***em***). De forma que, si definimos un punto de partida, podemos establecer el resto de tamaños de forma relativa al original, y evitamos problemas de herencia en cascada que, a veces, son difíciles de detectar.

Eso nos permite jugar con facilidad con las proporciones. Por ejemplo, podemos definir lo siguiente en una hoja de estilo:

```
html {
    font-size: 1em;
}

p {
    font-size: 0.7rem;
```

```
}

div {
    font-size: 0.9rem;
}
```

Si, posteriormente, hay elementos **<*p*>** dentro de un elemento **<*div*>**, el valor para el tamaño de fuente será idéntico a los que se encuentren ubicados fuera de un **<*div*>**.

Características especiales para las fuentes *OpenType*

La sección 6.12 de la especificación del módulo de fuentes CSS de nivel 3[60], permite especificar la sustitución de glifos y el posicionamiento de fuentes que incluyen las características de diseño **Microsoft OpenType**, utilizando la propiedad ***font-feature-settings***, cuyo patrón sintáctico es el siguiente:

font-feature-settings: normal **|** *feature-tag-value* [, *feature-tag-value*]

donde *feature-tag-value* se refiere a una lista separada por comas de diferentes características definidas por el documento "*OpenType layout feature tag registry*"[61], que dispone de más de 100 posibles valores de ajuste para las fuentes de esta clase, lo que capacita un número enorme de posibilidades de diseño.

El propio sitio de MSDN, recomienda su documento "*Cómo mejorar el diseño de tipo del sitio web con CSS3*"[62], para una referencia mucho más extensa sobre estas posibilidades.

Como un ejemplo simple consideremos esta regla:

[60] http://www.w3.org/TR/css3-fonts/

[61] http://www.microsoft.com/typography/otspec/featurelist.htm

[62] http://msdn.microsoft.com/es-es/library/ie/gg699339(v=vs.85).aspx

```
font-feature-settings: "dlig" 1;
```

El resultado es activar (el 1 a continuación activa, mientras el 0 desactiva) una característica de ligadura sobre la fuente donde se aplique. Si se quieren utilizar múltiples efectos sobre la misma fuente, podemos expresarlo separado por comas:

```
font-feature-settings: "dlig" 1, "ss02" 1, "case" 1;
```

El problema del ajuste del tamaño de las fuentes

Como sabe el lector, las fuentes se miden basándose en la altura de las letras mayúsculas, aunque sean las minúsculas las que muchas veces sufran más variaciones entre una y otra fuente. Esto resulta muy evidente en las implementaciones de algunos tipos de letra en diversas plataformas, o cuando, al no existir la fuente indicada por nosotros, el sistema utiliza las fuentes predeterminadas.

La propiedad ***font-size-adjust***, permite precisamente, establecer el tamaño de la fuente en relación a las letras minúsculas, solventando en buena parte este problema. El parámetro permite indicar un valor porcentual respecto a la altura de las letras mayúsculas, de forma que si le indicamos ***font-size-adjust = .5*** y usamos una letra de 22 píxeles, obligaremos a que las letras minúsculas tengan una altura de 11 píxeles.

Por consiguiente, si usamos un valor de 1, como en este código:

```
p:nth-of-type(2) {
    font-family:Fuente2;
    font-size-adjust:1;
}
```

Mayúsculas y minúsculas tendrán la misma altura, produciendo, de hecho, un escalado de la fuente, que presentaría una salida como la de la figura 33:

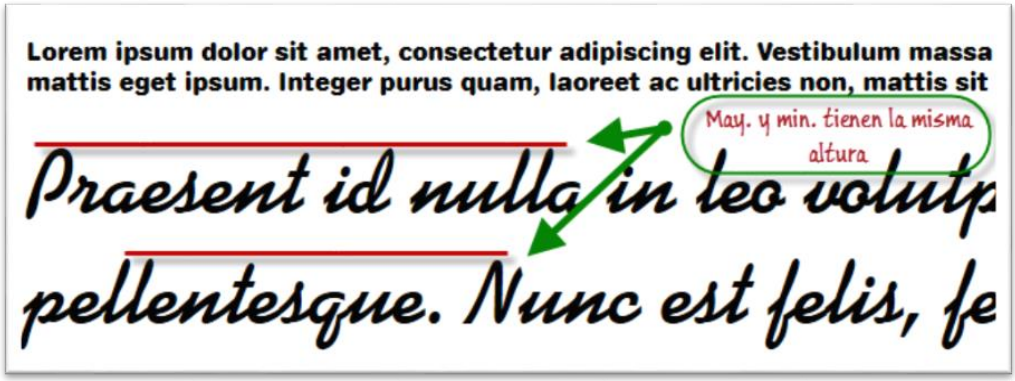


Fig. 33: Ajuste de letras minúsculas de una fuente mediante font-adjust

Texto con sombra

Otro efecto visual muy esperado era el de poder aplicar ideas similares a las de la sombra de las cajas al propio texto. El estándar contempla la propiedad ***text-shadow***, muy fácil de implementar indicando las propiedades en forma muy similar a como hemos procedido con la sombra de los rectángulos.

La forma de utilizarlo es muy simple: se indica el color de la sombra, y los desplazamientos correspondientes en forma similar al de las cajas mediante un código similar al siguiente:

```
h1 { color: black; text-shadow: #87CEEB 1px 3px; }
```

Y en la salida, al aplicarse al texto del ejemplo anterior, tendremos el siguiente resultado:

Fig. 34: Texto con sombra

Naturalmente, jugando con los valores de desplazamiento, resulta sencillo obtener otros efectos como el de letras grabadas, distorsiones, letras con bordes, etc. Normalmente, cada parámetro, según sus valores y signo, aportará un conjunto de efectos diferente. Por ejemplo, para ver las letras como si estuvieran grabadas, podemos usar esta combinación:

```
h2
{
    color: #000;
    background-color:#e9e8e8;
    font-family:"Special Fonts";
    font-size:50pt;
    text-shadow: 0px -2px -1px #374683;
}
```

Que, aplicada al título de este documento de prueba, nos muestra una salida como la de la Figura 35:

Fig. 35: Texto con efecto de grabado

Téngase en cuenta, además, que existe la posibilidad de asignar varias sombras al mismo elemento, separando por comas las distintas definiciones. Por ejemplo, podemos imaginarnos dos variantes del mismo título en el que jugamos con el color de fondo, el de la letra y las sombras:

```
h2
{
    color: #000;
    background-color:#e9e8e8;
    font-family:"Special Fonts";
    font-size:50pt;
    text-shadow:
    0 -2px 3px #FFF, 0 -4px 3px #AAA, 0 -6px 6px #666;
}
```

En este caso, al establecer tres sombras de distintos colores, se obtiene un relieve como el que vemos en la Figura 36:

Fig. 36: Efecto de varias sombras sobre el título.

Y también podemos optar por la opción inversa respecto a los colores de sombras, fondo y primer plano, como en el código siguiente:

```
h2
{
background-color:#e9e8e8;
font-size:50pt;
color: #FFF;
text-shadow: 0 2px rgba(0,0,0,0.4),0 4px rgba(0,0,0,0.4),
             0 6px rgba(0,0,0,0.4); }
```

Lo que nos genera, igualmente, una salida con contraste totalmente distinto (Ver Figura 37)

Figura 37. Efecto de sombras inverso al anterior actuando sobre el mismo elemento

Vemos, pues que el conjunto de posibles efectos con el texto es muy grande, especialmente cuando mezclamos estas características con las de definición de colores (primer plano, sombra, fondo, etc.).

text-overflow

Lo mismo sucede con otras propuestas relativas al formato de texto. Por ejemplo, ***text-overflow*** permite especificar qué se debe hacer con un texto que no cabe en la caja correspondiente. Los valores posibles, ***ellipsis*** y ***clip***, establecen si se muestran puntos suspensivos en lugar de una palabra truncada o se permite ver ésta tal y como aparezca en las condiciones de la interfaz.

Y hay más opciones interesantes en el tratamiento de texto, como es la posibilidad de disponer el texto en columnas que veremos a continuación.

Modificaciones que afectan a la estructura del documento

Hasta aquí, hemos visto los cambios que afectan a las unidades, al modelo de caja y sus efectos especiales, y a la presentación del texto. Pero hay otras novedades que se aplican directamente a la estructura del

contenido del documento y cómo éste se presenta respecto a otros, como las columnas, las exclusiones, las cajas de contenido flexible, el diseño de cuadrículas, las regiones, etc.

Hay que notar, que algunas de estas propuestas son de muy reciente implantación, por lo que –si tenemos que soportar navegadores antiguos- es más que probable que tengamos que recurrir a algún *fallback* para ofrecer la misma funcionalidad.

Columnas

Mención aparte merece el nuevo tratamiento de columnas que es posible definir ahora, aplicable a un texto cualquiera. La especificación lo ha separado en un módulo independiente, que ya alcanzó en abril de 2011 el estado de "*Candidate Recommendation*"[63].

Seguiremos trabajando con el último ejemplo, esta vez para implementar la especificación de múltiples columnas para la estructura del texto. Para ello, disponemos de un conjunto de nuevas propiedades. Las más importantes son las siguientes:

- ***column-count: [nº entero]*** Establece el número de columnas
- ***column-width: [double]*** Establece la anchura de una columna
- ***column-fill: [auto |balance]*** Relleno secuencial o ponderando la cantidad de texto
- ***column-gap [double]*** Indica el valor de la separación entre columnas
- ***column-rule [length border-style color]*** Dibuja una línea (similar a un borde) de forma equidistante entre las columnas. Se puede establecer indicando los 3 valores posibles (longitud, estilo de borde y color), o utilizar las sub-propiedades asociadas con él: ***column-rule-width***, ***column-rule-style*** y ***column-rule-color***.

[63] http://www.w3.org/TR/css3-multicol/

- ***column-span: [1 | all]*** Permite que un elemento se expanda más de una columna
- ***break-**** Permite establecer el comportamiento para elementos distintos del texto habitual que no pueden ser divididos entre columnas, tales como listas, subtítulos, etc. ***Break*** permite definir el punto en que termina una columna y comienza la siguiente, y admite 3 valores: ***break-after***, ***break-before*** y ***break-inside***. Los 3 admiten como valores posibles las palabras reservadas ***auto*** (predeterminado) y ***avoid*** (impide la ruptura, antes o después), y –además- los 2 primeros también admiten la palabra ***column*** (que se comporta de la forma opuesta, forzando una ruptura antes o después del elemento al que se aplique).

Veamos algunos ejemplos del funcionamiento de estas propiedades y comenzamos con la más sencilla: cómo establecer el número de columnas que se desean para un texto dado.

Si para el párrafo que contiene el texto "*Lorem Ipsum*" del ejemplo anterior definimos una división en 2 columnas con la sintaxis `column-count: 2;` obtendremos una salida como la de la figura 17 (las variantes para los navegadores basados en Mozilla y Webkit son `-moz-column-count: 2;` y `-webkit-column-count: 2;` respectivamente.

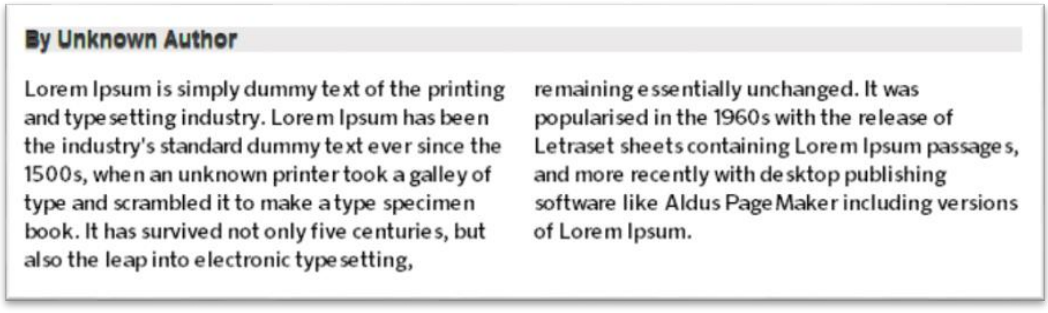


Fig. 38: División simple en dos columnas del texto de ejemplo.

Una posibilidad más flexible consiste en utilizar la segunda opción, que

permite establecer la anchura deseada para las columnas, independientemente del navegador y la experiencia de usuario, como establece el siguiente código:

```
-moz-column-width:120px;
-webkit-column-width:120px;
column-width:120px;
```

Las versiones IE10/IE11 soportan casi todas ellas. La salida correspondiente para un tamaño similar del navegador, nos ofrece una división en 4 columnas, como vemos en la figura 39.

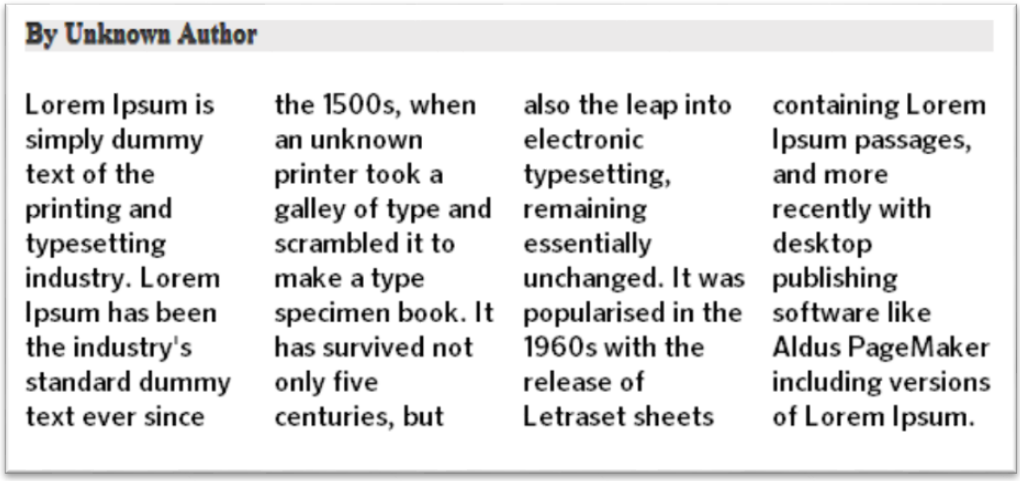


Fig 39: Al definir las columnas por su anchura, el número total de ellas es dinámico, cuando se modifica la anchura del elemento.

Naturalmente, también podemos combinar ambas propiedades, estableciendo el número de columnas y el ancho de cada una.

En el ejemplo, siguiente, vamos a utilizar también un separador de columnas y unas reglas para indicar que el separador será una línea doble, color azul, y con un ancho de separación total de 12 píxeles:

```
column-width:120px;
column-gap:4em;
column-rule: 12px double blue;
```

Lo que nos ofrecerá una salida como la de la figura 40:

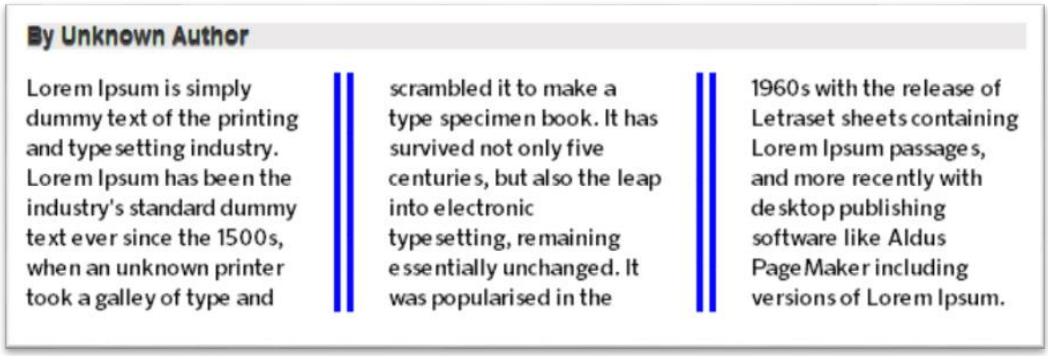


Fig. 40: Salida del código anterior en IE10

Hay todavía más posibilidades relacionadas con las columnas y el soporte –aunque no totalmente completo aún- es bastante homogéneo en todos los navegadores analizados.

Exclusiones

Esta opción es una novedad en IE10, y consiste en poder ajustar el texto de manera que rodee completamente un elemento dado, sea de la naturaleza que sea. Como se indica en el blog de IE10: "En lugar de limitar los elementos a flotar a la izquierda o a la derecha en relación con su posición en el flujo del documento, las exclusiones CSS pueden posicionarse a una distancia especificada desde las partes superior, inferior, izquierda o derecha de un bloque que las contenga, y siguen formando parte del flujo del documento".

No se trata de una extensión privada de IE10, sino de parte del estándar, ya que pertenece al borrador oficial "*CSS Exclusions and Shapes Module Level 3*"[64], de junio/2012. De ahí que, por el momento, para ver esta característica, usemos las extensiones propias de IE (***-ms***).

[64] http://dev.w3.org/csswg/css3-exclusions/

La documentación oficial del MSDN recomienda tener en cuenta los siguientes términos, a la hora de trabajar con exclusiones:

- **Elemento de exclusión:** La regla **-ms-wrap-side** convierte un elemento en un *elemento de exclusión* cuando tiene un valor calculado que no sea **auto**. Esto especifica que el elemento de exclusión puede ubicarse de diferentes maneras, y que el contenido en línea se ajustará en torno a la exclusión de una forma similar a la que se ajusta en torno a los elementos flotantes.
- **Área de exclusión:** El *área de exclusión* es el área que se usa para excluir contenido alrededor de una exclusión. Decimos que el área de exclusión es equivalente al cuadro de borde para elementos con una regla ***float:*** **none**, y el cuadro de margen para los demás. De momento, siempre será un rectángulo.
- **Área de contenido:** El *área de contenido* es el área que se usa para el contenido de flujo en línea de un elemento. De manera predeterminada es equivalente al cuadro de contenido.
- **Contexto de ajuste:** El *contexto de ajuste* de un elemento es una colección de áreas de exclusión. Un elemento ajusta su contenido de flujo en línea en el área que resulta de restar el contexto de ajuste de su área de contenido. Un elemento hereda el contexto de ajuste del bloque que lo contiene a menos que se restablezca específicamente con la propiedad **-ms-flow-wrap** (permite establecer excepciones).

Los valores posibles para **–ms-wrap-side**, son los siguientes:

- **auto:** En el caso de elementos flotantes, se crea una exclusión; en el caso de los demás elementos, no se crea una exclusión.
- **both:** El contenido de flujo en línea puede distribuirse por todos los lados de la exclusión.

- **left:** El contenido de flujo en línea puede ajustarse al borde izquierdo del área de exclusión, pero debe dejar vacía el área situada a la derecha del área de exclusión.
- **right:** El contenido de flujo en línea puede ajustarse al lado derecho del área de exclusión, pero debe dejar vacía el área situada a la izquierda del área de exclusión.
- **máximum:** El contenido de flujo en línea puede ajustarse al lado de la exclusión que tenga más espacio disponible para la línea dada y debe dejar vacío el otro lado de la exclusión.
- **clear:** El contenido de flujo en línea puede ajustarse encima y debajo de la exclusión y debe dejar vacías las áreas situadas en los bordes de inicio y fin del cuadro de exclusión.

Además, podemos utilizar la regla ***–ms-wrap-margin***, para especificar un margen que se usa para desplazar la forma interna de ajuste de otras formas.

Vamos a utilizar la página de ejemplo, para aplicarle el siguiente código de estilos (obsérvese que se utilizan también otras reglas tipo ***–ms-grid***, para la ubicación de los elementos).

```
.contenedor {
    font-size: small;
    display: -ms-grid;
      height: 80%;
      overflow: hidden;
    -ms-grid-columns: 1fr 1fr 1fr;
    -ms-grid-rows: 1fr 1fr 1fr;
}

.exclusion {
    -ms-grid-row: 2;
    -ms-grid-column: 2;
    background-color: lime;
    -ms-wrap-flow: both;
    padding: 10px;
```

```
    -ms-wrap-margin: 15px;
     z-index: 1;
}

.texto {
    -ms-grid-row: 1;
    -ms-grid-column: 1;
    -ms-grid-column-span: 3;
    -ms-grid-row-span: 3;
}
```

Tenga presente el lector que en el momento actual que las definiciones del estándar no llevan prefijo de vendedor, aunque –por el momento- solo lo soporta IE. Aplicamos estos estilos al siguiente código HTML:

```
<div class="body">
    <h1>Ejemplo de Exclusiones CSS3</h1>
    <div class="contenedor">
        <div class="exclusion">
            Exclusión ubicada en fila 2, columna 2
        </div>
        <div class="texto">
            <p>Lorem ipsum dolor sit amet, consecte
                feugiat. Mauris volutpat dui odio,
                amet dignissim eros. Fusce et metus
                aptent taciti sociosqu ad litora to
                Suspendisse in nulla sed libero fau
                eros orci. Duis interdum lobortis t
                semper quam non augue scelerisque e
                mollis massa ultrices id ultricies
            </p>
            <p>Praesent nec mi arcu. Cras eu nulla
                vitae pharetra non, faucibus nec di
            </p>
       </div>
   </div>
</div>
```

Con lo que obtenemos la siguiente salida en IE10/11:

Fig. 41: Ejemplo de zonas de exclusión en IE10/11

El diseño de caja flexible (*Flexbox*)

Se trata de una propuesta implementada en IE10, basada en el borrador "*CSS Flexible Box Layout Module*"[65] (junio/2012), de la W3C, que está siendo construido por técnicos de Google, Microsoft, Mozilla y Opera.

La documentación oficial, explica que esta propuesta "*tiene en cuenta el espacio disponible cuando se definen las dimensiones de la caja, lo que permite utilizar tamaños y posicionamientos relativos. Por ejemplo, podemos asegurarnos de que el espacio en blanco extra de la ventana de un explorador se distribuya homogéneamente según el tamaño de varios elementos secundarios, y que estos elementos secundarios estén centrados*

[65] http://www.w3.org/TR/css3-flexbox/

en el medio del bloque que los contiene".

Esto se basa en un contenedor *flexbox*. Para ello, establecemos la propiedad ***display*** de un elemento en ***-ms-flexbox*** (para un contenedor *flexbox* de nivel de bloque) o ***-ms-inline-flexbox*** (para un contenedor *flexbox* alineado). El siguiente ejemplo crea un contenedor *flexbox* de nivel de bloque dentro del elemento que tiene un identificador "*cajaFlexible*":

```
#cajaFlexible {
  display: -ms-flexbox;
  background-color:#cce0ee;
  border: Navy;
}
```

Al crear un contenedor *flexbox*, también se puede establecer su orientación—es decir, indicar si su contenido se mostrará de derecha a izquierda, de izquierda a derecha, de arriba a abajo o de abajo a arriba. Para hacerlo, se usa la propiedad ***-ms-flex-direction***, que admite los valores ***row, column, row-reverse, column-reverse*** e ***inherit.***

Y, en cuanto a su alineación, se dispone de la propiedad ***-ms-flex-align***, teniendo en cuenta que se establecerá de forma perpendicular al eje definido por ***–ms-flex-direction*** y que dispone de los siguientes valores posibles: ***start, end, center, stretch*** y ***baseline.***

Otras propiedades completan la oferta. A falta de un mayor nivel de implementación, podemos ver un pequeño ejemplo de funcionamiento, utilizando el código siguiente para formatear cuatro elementos *** *** de forma que aparezcan en un orden diferente al original, y empotrados en un *flexbox* previamente definido.

```
<body>
    <style type="text/css">
        #cajaFlexible
        {
            display: -ms-flexbox;
            color: white;
            font-size: 48px;
            font-family: "Segoe UI"
```

```
            text-align: left;
            height: 200px;
            border: none;
        }

        #hijo1
        {
            -ms-flex-order: 1;
            background: #43e000;
            padding: 20px;
        }

        #hijo2
        {
            -ms-flex-order: 0;
            background: #166aff;
            padding: 20px;
        }

        #hijo3
        {
            -ms-flex-order: 1;
            background:#808080;
            padding: 20px;
        }

        #hijo4
        {
            background:#ff006e;
            padding: 20px;
        }
    </style>
    <div id="cajaFlexible">
        <div id="hijo1">1</div>
        <div id="hijo2">2</div>
        <div id="hijo3">3</div>
        <div id="hijo4">4</div>
    </div>

</body>
```

Lo que genera una salida reordenada (propiedad **–ms-flex-order**) dentro de un contenedor donde no se establece una dirección específica (y por lo tanto, se utiliza la horizontal). Esto nos genera una salida como la de la figura adjunta:

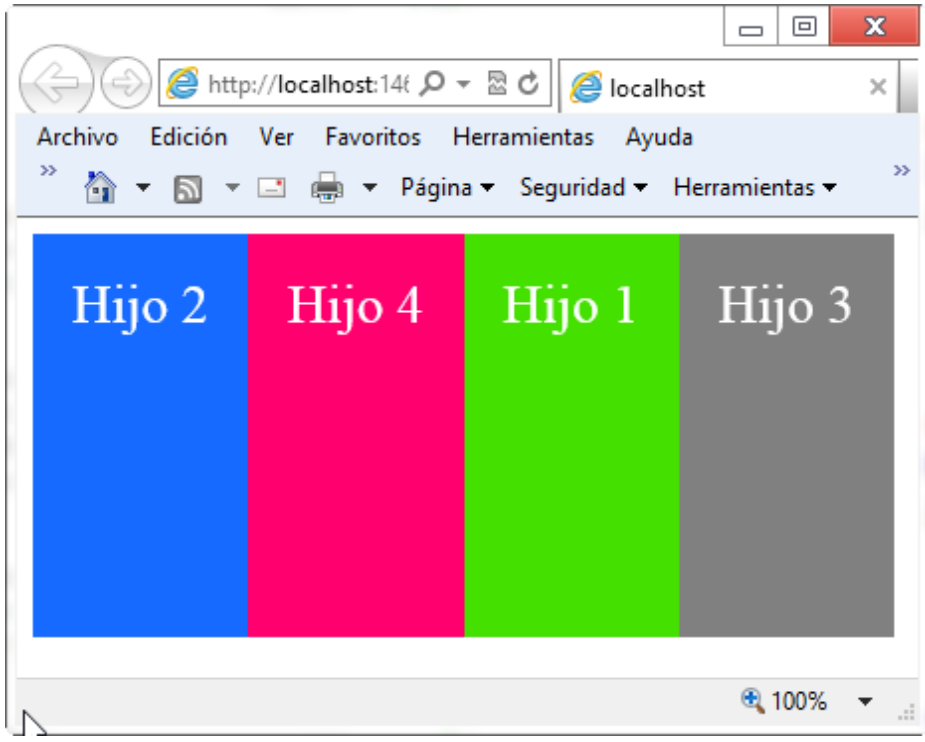


Fig. 42: Flexbox en funcionamiento en IE10

Y hay más novedades en este tipo de diseños flexibles, como vamos a ver a continuación en lo que se denomina "Diseño de Cuadrícula".

El diseño de cuadrícula

Al igual que *Flexbox*, la cuadrícula permite lograr una mayor fluidez de diseño de lo que es posible obtener con el posicionamiento mediante elementos flotantes o un *script*. Permite dividir el espacio por regiones principales y definir la relación entre las partes de un control de HTML en términos de tamaño, ubicación y nivel. No hay necesidad de diseños fijos que no puedan aprovechar la superficie de los exploradores.

La especificación en que se basa es la "*CSS Grid Layout*"[66], cuyo *Working Draft* activo es de Marzo de 2012.

La ventaja principal de esta estructura es que permite escenarios que no son posibles con las tablas de HTML, ya que permite alinear los elementos en filas o columnas, pero, en sí misma, no tiene una estructura de contenido (como las tablas). Si la utilizamos junto a las directivas ***@media*** (ver apartado siguiente), se puede conseguir que el diseño se adapte perfectamente a los cambios de factor de forma, orientación, etc.

Creación de un elemento tipo *Grid*

Para crear uno de estos elementos, basta con asignar una regla ***display:-ms-grid*** para una cuadrícula de nivel de bloque, o bien ***display:-ms-inline-grid*** para una cuadrícula en línea. Una vez hecho esto, podemos determinar el tamaño de las columnas y filas mediante las siguientes propiedades:

Propiedad	Descripción
-ms-grid-columns	Determina el ancho de cada columna dentro de la cuadrícula. Cada columna está delimitada con un espacio.
-ms-grid-rows	Determina el alto de cada fila dentro de la cuadrícula. Cada fila está delimitada con un espacio.

El ajuste del tamaño de las columnas y filas, (a las que se llama *pistas* en este contexto), se realiza mediante las unidades conocidas, porcentajes de la anchura del objeto, unidades de fracción (***fr***), o las palabras reservadas ***auto***, ***min-content*** y ***max-content.***

Como siempre, partir del ejemplo oficial nos puede dar pistas más claras sobre la forma de utilizar estas cuadrículas. Supongamos una cuadrícula

[66] http://www.w3.org/TR/css3-grid-layout/

que definimos sobre un elemento llamado ***cuadricula***, con estos valores:

```
#cuadricula {
  display: -ms-grid;
  background: gray;
  border: blue;
  -ms-grid-columns: auto 100px 1fr 2fr;
  -ms-grid-rows: 50px 5em auto;
}
```

La interpretación que el navegador hará de esa definición será la siguiente:

- Columna 1 (palabra clave **auto**): la columna se ajusta al contenido interno.
- Columna 2 ("100px"): la columna mide 100 píxeles de ancho.
- Columna 3 ("1fr"): la columna ocupa una unidad de fracción del espacio restante.
- Columna 4 ("2fr"): la columna ocupa dos unidades de fracción del espacio restante.

Y cada fila se visualiza de la siguiente manera:

- Fila 1 ("50px"): la fila mide 50 píxeles de alto.
- Fila 2 ("5em"): la fila mide 5em de alto.
- Fila 3 (palabra clave auto): la fila se ajusta al contenido interno.

La numeración de pistas es un índice basado en 1, donde el 1 es el valor predeterminado. Si tenemos en cuenta la cuadrícula antes declarada, podemos considerar los siguientes dos selectores de identificación:

```
#item1 {
  color:yellow;
  background: maroon;
```

```
  border: orange solid 1px;
  -ms-grid-row: 1;
  -ms-grid-column: 1;
}
#item2 {
  background: lightgray;
  border: red solid 1px;
  -ms-grid-row: 2;
  -ms-grid-column: 2;
}
```

Con esas definiciones, si disponemos de un elemento *** *** y añadimos un par de elementos internos, de nombres "*item1*" e "*item2*", mediante este sencillo código:

```
<div id="cuadricula">
    <div id="item1">Elemento 1</div>
    <div id="item2">Elemento 2</div>
</div>
```

Al aplicar los estilos obtendremos la siguiente salida por pantalla:

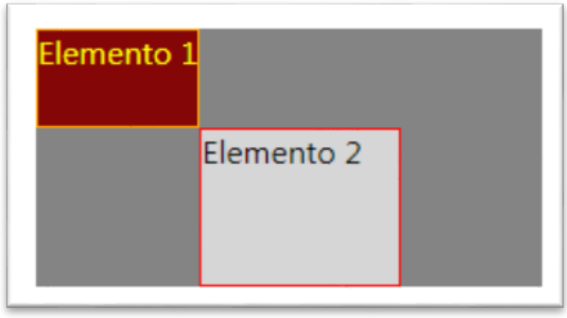


Fig. 43: Salida del diseño de cuadrícula del código anterior en IE10/11

Además, existen dos propiedades que nos permiten colocar los elementos de una cuadrícula en cualquiera de las celdas formadas por las columnas y filas del ***Grid***. A tal efecto disponemos de:

- **-ms-grid-column-align**: que determina la alineación horizontal del elemento dentro de la columna de la cuadrícula (valores posibles son **center**, **end**, **start** y **stretch**).
- **-ms-grid-row-align**, que permite controlar la alineación vertical del elemento dentro de la fila de la cuadrícula, y que tiene los mismos valores posibles que el anterior.

Por ejemplo, si le añadimos un elemento más a nuestra cuadrícula anterior de forma que tengamos 3 elementos:

```
<div id="cuadricula">
    <div id="item1">Elemento 1</div>
    <div id="item2">Elemento 2</div>
    <div id="item3">Elemento 3</div>
</div>
```

Podemos añadir un nuevo selector para el elemento 3 de la siguiente forma:

```
#item3 {
  background: orange;
  border: maroon solid 1px;
  -ms-grid-row: 3;
  -ms-grid-column: 3;
}
```

Y modificar el segundo de manera que aparezca en otra posición:

```
#item2 {
  background: lightgray;
  border: red solid 1px;
  -ms-grid-row: 2;
  -ms-grid-column: 2;
  width: 40px;
  height: 30px;
  -ms-grid-column-align: end;
  -ms-grid-row-align: center;
}
```

Tras esos cambios, se producirá la salida de la Figura 44:

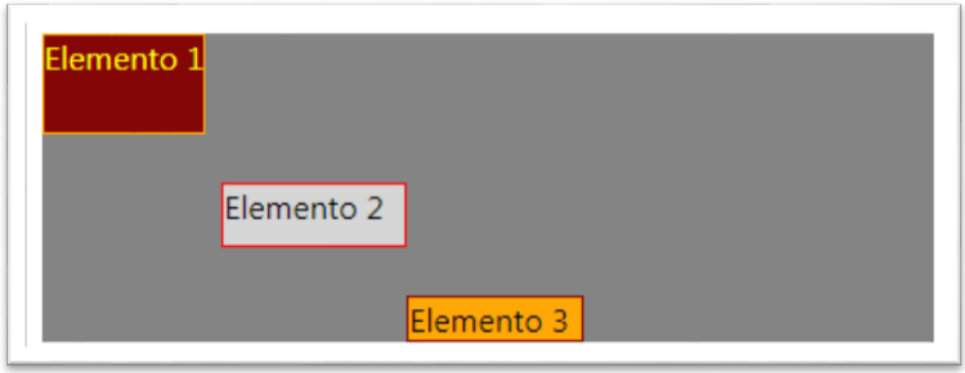


Fig. 44: Salida en IE10/11 del código anterior.

Finalmente, con las propiedades ***-ms-grid-column-span*** y ***-ms-grid-row-span*** podemos lograr que los elementos de cuadrícula se extiendan a varias columnas o filas. El valor predeterminado es "1".

Regiones

Otra de las novedades más interesantes respecto al posicionamiento de los elementos de una página, lo aporta el sub-estándar CSS3 Regions[67], que ya se encuentra en el estado "*Working Draft*", con fecha de Mayo/2012.

Mediante este concepto, podemos recibir un origen HTML que incluya texto de marcado e imágenes, y fragmentarlo dentro de varios contenedores vacíos que hayamos dispuesto a tal efecto, que contendrán definiciones de estilo para ubicación y disposición visual.

Entre otras ventajas, esto permite que un "*streaming*" continuo pueda ser ubicado correctamente o reorganizado, digamos, para favorecer su presentación en un dispositivo de tipo *tablet*. Otro posible uso es el de

[67] http://www.w3.org/TR/css3-regions/

simular composiciones visuales similares a las de las revistas o periódicos, donde múltiples regiones del mismo flujo de contenido, como texto, vídeos, imágenes, etc., se formatean alrededor de otro contenido que no está relacionado con él, como pueden ser anuncios publicitarios o historias no relacionadas.

Supongamos un documento de servidor con su propio DOM y su propio estilo CSS (representado en el esquema de la figura 45 por el fichero "*Contenido.html*"). A continuación necesitamos otro fichero que sirva de *host* o de página maestra ("*Master.html*", en el diagrama), si se quiere, para alojar y ubicar el contenido en sus distintas regiones, identificando cada zona para la ubicación de cada partes alojada. Esto no afectará al modelo DOM de la página de contenido.

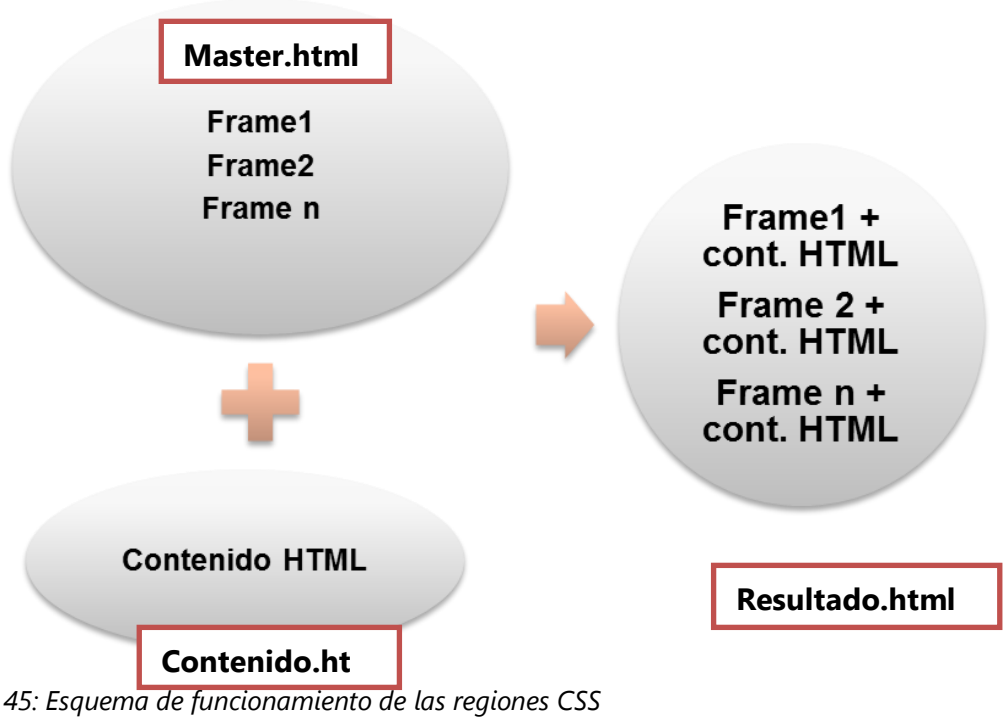


Fig. 45: Esquema de funcionamiento de las regiones CSS

Téngase en cuenta que el flujo dinámico puede significar que un mismo marco de contenido (*frame*) pueda ir alojando diversas informaciones

que le lleguen periódicamente, por ejemplo.

Hay varias propiedades nuevas que posibilitan esta operativa: ***–ms-flow-from*** y ***-ms-flow-into***, que indican, respectivamente, el origen y el destino y de los contenidos de un elemento.

Esto, en la práctica, puede combinarse con los diseños de cuadrícula vistos en el punto anterior, para conseguir que distintos fragmentos de código "pasen" de un contenedor a otro dinámicamente, ante una acción cualquiera del usuario, que simplemente se limitará a modificar los destinos de cada fragmento.

El código oficial del sitio de desarrollo de IE10 (único navegador que lo implementa en este momento), tiene un aspecto similar a este:

```
<iframe src="text_content.htm"
    style="-ms-flow-into: MyFlow"></iframe>
```

Y los contenedores de "*stream* HTML":

```
<div style="-ms-grid-column: 2; -ms-grid-row: 2; -ms-
flow-from: MyFlow;">
</div>

<div style="-ms-grid-column: 2; -ms-grid-row: 4; -ms-
flow-from: MyFlow;">
</div>

<div style="-ms-grid-column: 2; -ms-grid-row: 8; -ms-
flow-from: MyFlow;">
</div>

<div style="-ms-grid-column: 4; -ms-grid-row: 2; -ms-
flow-from: MyFlow;">
</div>

<div style="-ms-grid-column: 4; -ms-grid-row: 6; -ms-
flow-from: MyFlow;">
</div>
```

Visto en IE10/11, observaremos diversas regiones de contenido divididas del resto, pero que pueden ser fácilmente conmutables, como podemos ver en las dos instantáneas de la figura 46, donde la primera recoge una posible posición inicial, y la segunda es el resultado de haber pulsado el enlace "*Page 2*", que no lleva realmente a una nueva página, sino que ubica el siguiente conjunto de contenidos dentro de los contenedores existentes.

Como puede entender el lector, la operativa es especialmente conveniente en ciertos contextos, como los *tablets*, o la propia interfaz de usuario de Windows 8.

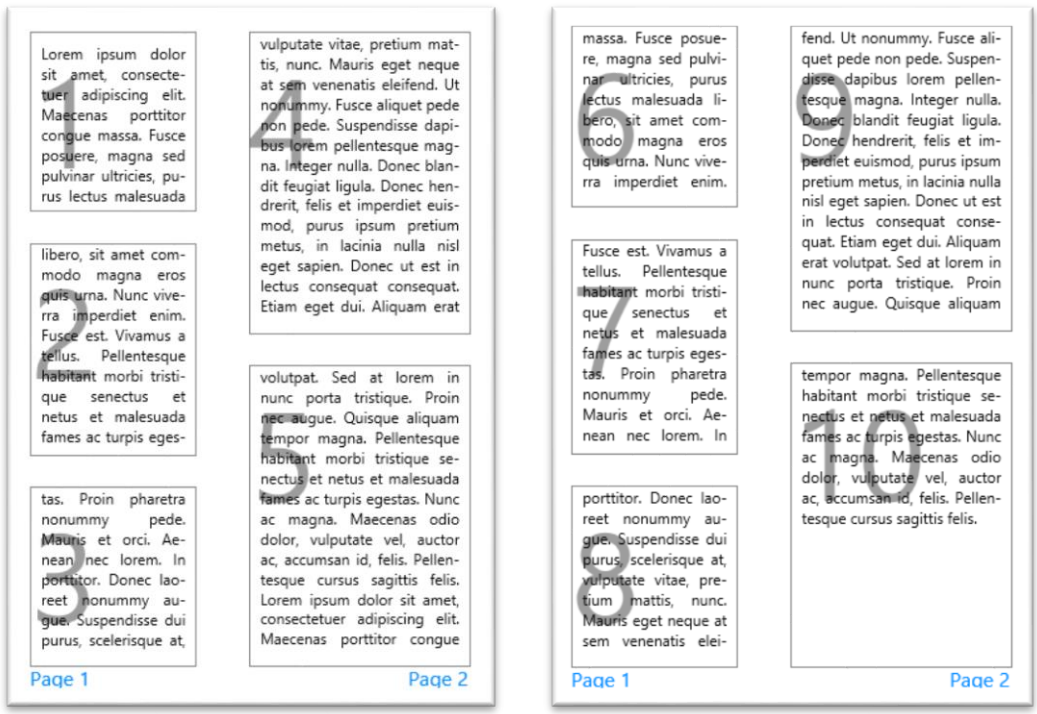


Fig. 46. El ejemplo oficial del sitio de desarrollo de IE10, mostrando dos vistas de la misma página, cuyo contenido ha "fluido" a través de sus contenedores.

Adviértase que las zonas de flujo de contenido permanecen ubicadas en el mismo sitio y con su tamaño inalterable según las definiciones de sus hojas de estilo. Es el HTML que sirve de fuente el que se fragmenta, se

ubica en las distintas cajas, y fluye dinámicamente.

Media Queries: Adaptación a dispositivos

Se trata de directivas (al estilo de lo que hemos visto antes con ***@font-face***), que realizan una comprobación inicial que nos permite determinar el tipo de dispositivo en el que se muestra la página, de forma que podamos adaptar el contenido a las características del contexto.

En esta versión, consisten en una serie de mejoras relacionadas con la regla (directiva) ***@media***, ya presente en versiones anteriores (genéricamente, ***media types***), que permite una sintaxis de comprobación que funciona realmente como una sentencia ***if***, de forma que aplican o no las sentencias del bloque si la condición se cumple. Por ejemplo, podemos indicar un cierto estilo aplicable a las salidas por impresora, o la disposición visual en un teléfono o TV.

Mientras que en la versión anterior, la lista completa de ***media types*** incluía '***aural'***, '***braille'***, '***handheld'***, '***print'***, '***projection'***, '***screen'***, '***tty'*** y '***tv'***, en la nueva "***aural***", se considera obsoleta, debido al nuevo soporte de atributos WAI-ARIA, y se añaden los tipos "***embossed***", y "***speech***"[68].

Por tanto, mediante la regla **@media** y las nuevas palabras reservadas, podemos establecer indicaciones especiales para un dispositivo de salida, cuando sabemos que el soporte es adecuado. Por ejemplo, para indicar una regla para los dispositivos móviles, podemos utilizar los operadores ***only*** y ***and***, junto a la indicación de orientación para definir cómo queremos que se muestren los elementos de una lista:

[68] Respecto a este punto, existe una especificación concreta, con su propio módulo, "*CSS3 Speech Module*", disponible en http://www.w3.org/TR/css3-speech/ que contiene indicaciones de tipo aural para indicar el comportamiento de los dispositivos de síntesis de voz.

```
ul { overflow: hidden; }
li { float: left; }
@media only screen and (orientation: portrait)
{
    li { float: none; }
}
```

Donde indicamos al agente de usuario que –solo en el caso de que el dispositivo sea una pantalla y utilice orientación vertical (como en los móviles) debe de aplicar la condición ***float: none***.

También podemos indicar condiciones para el tamaño visual del dispositivo de salida, o exigir unos tamaños mínimos o máximos antes de aplicar una regla. Los prototipos de código adoptarían una sintaxis general similares a los siguientes:

```
@media media and (device-height:value) { ... }
@media media and (max-device-height:value) { ... }
@media media and (min-device-height:value) { ... }
```

Carga condicional

Naturalmente, en aplicaciones de la vida real, probablemente utilizaremos más de una hoja de estilo que podemos cargar a petición o tras comprobar que se cumplen ciertas condiciones. Para ello, es posible codificar la carga de una u otra hoja de estilo en función de unas condiciones establecidas:

```
<link href="Inicial.css" rel="stylesheet" media="screen">
<link href="Extendido.css" rel="stylesheet" media="screen
and (min-device-width: 420px)">
```

En el caso de que queramos cubrir la posibilidad de que se trate de versiones de IE anteriores a la 9, podemos usar –adicionalmente- la siguiente sintaxis, que IE comprende perfectamente:

```
<!--[if lt IE 9]>
  <link href="Extendido.css" rel="stylesheet"
media="screen”>
<![endif]-->
```

Donde le decimos que si la versión de IE es menor que la 9, utilice la hoja "Extendido.css" para salidas por pantalla.

Igualmente, es posible establecer condiciones más específicas, como la relación de aspecto (***aspect ratio)*** que se dan en conjuntos de dispositivos similares (como puede suceder con los móviles o ciertos *tablets*). Disponemos de dos opciones relacionadas cuyo código genérico podría tener un aspecto como este:

```
@media [medio] and (aspect-ratio: horizontal/vertical)
{…}
@media [medio] and (device-aspect-ratio:
horizontal/vertical) {…}
```

Y lo mismo cabe decir de otros aspectos visuales como el "*pixel ratio*" que podríamos manejar comprobándolo mediante indicaciones que tienen un prototipo de código como este:

```
@media [medio] and (-moz-device-pixel-ratio: [número])
{…}
```

Para una documentación exhaustiva y actualizada del estado del estándar respecto a las consultas multimedia, recomendamos el documento oficial "*Media Queries*"[69], que ha sido actualizado en Junio/2012.

Igual que sucede con las reglas que hemos visto en este capítulo, las directivas también disponen de extensiones de navegador, y Microsoft ha creado algunas opciones de interés que vemos a continuación.

[69] http://www.w3.org/TR/css3-mediaqueries/

Directivas CSS como extensiones de un navegador

Con el objeto de soportar una diversidad de contextos de visualización, Microsoft incorpora la directiva ***@-ms-viewport***. Se basa en la regla ***@viewport***, que está definida en la especificación *Adaptación de dispositivos CSS*, si bien, no se corresponde exactamente con la información definida en la especificación, ya que ambas (especificación e implementación en IE10) están en fase de construcción. Actualmente, esta parte del estándar está en fase de borrador de trabajo, y el lector puede encontrar la última versión en el documento "*CSS Device Adaptation*"[70], de junio/2012.

Suele utilizarse en conjunción con las directivas anteriores, con el propósito de afinar al máximo la presentación para una superficie. El ejemplo típico es el que ofrece el sitio de desarrollo de IE10, que hemos adaptado y modificado de la siguiente forma, para su visualización:

```
@media screen and (max-width: 400px) {
  @-ms-viewport { width: 320px; }
  /* CSS para visualización en 320px de ancho */
    p
    {
        font-family:'Segoe UI';
        font-size:xx-small;
    }
}
```

El código anterior solo actúa cuando el dispositivo adopta una anchura de 400px o menos. En ese caso, actúa una directiva que estrecha la visualización a 320px, y se le añade un formato de párrafo con un tipo y un tamaño de letra.

[70] http://dev.w3.org/csswg/css-device-adapt/

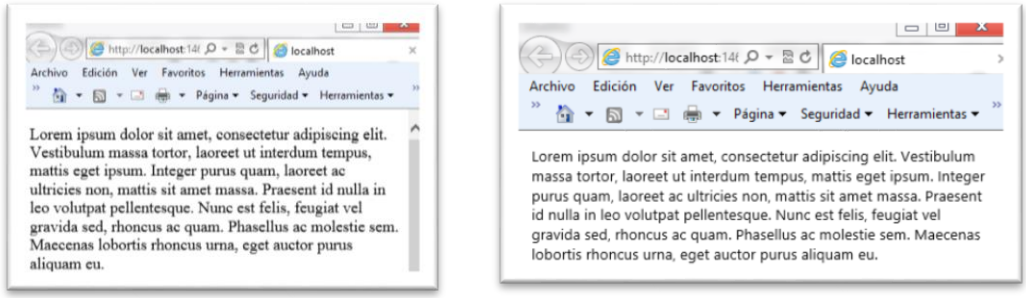


Fig. 47: Aplicación de la directiva @viewport en IE10

En tiempo de ejecución, la detección de la situación se produce debido a que éstas directivas son re-evaluadas cuando el DOM genera ciertos eventos de navegador, como ***resize*** o bien en la carga inicial (***load***). Como vemos a continuación, la figura de la izquierda muestra IE10 con un tamaño algo superior a 400px. En cuanto hacemos el tamaño ligeramente inferior a esa cantidad, se produce automáticamente el cambio, y pasamos a ver el contenido de la figura de la derecha.

De la misma forma, podemos aplicar estos principios para que la superficie se adapte y escale automáticamente, entre una vista horizontal y una vertical con código como el siguiente:

```
@media screen and (orientation: landscape) {
  @-ms-viewport {
    width: 1024px;
    height: 768px;
  }
  /* CSS para la vista horizontal */
}

@media screen and (orientation: portrait) {
  @-ms-viewport {
    width: 768px;
    height: 1024px;
  }
  /* CSS para la vista vertical */
}
```

Finalmente, si no se desea el ajuste de escala automático, podemos deshabilitarlo mediante un código tan simple como este:

```
@-ms-viewport { width: device-width; }
```

Donde hay que tener en cuenta que, al no estar la directiva incluida en ningún bloque ***@media***, se aplica para todos los casos, ajustándolo la anchura a la del dispositivo.

Elementos dinámicos

Todo lo visto aporta una enorme cantidad de novedades a las hojas de estilo, pero uno de los objetivos que perseguía HTML y sus tecnologías era el de no tener que recurrir a elementos empotrados en las páginas para poder suministrar efectos visuales, como son las transformaciones, las transiciones y las animaciones, tanto en 2D como en 3D.

El apartado de 3D está todavía en fase de desarrollo, por lo que no es posible verlo completamente en funcionamiento en todos los navegadores, pero sus correspondencias 2D (y las proyecciones bidimensionales del 2D), sí que lo están y gozan ya del favor popular, a la vista de que los navegadores actuales las soportan casi en su totalidad. Comenzamos hablando de las Transformaciones.

Transformaciones

Las transformaciones permiten definir reglas que modifican su presentación visual tanto en 2 como en 3 dimensiones. La especificación actual "*CSS Transforms*"[71], de Abril/2012, es un proyecto convergente que incluye las anteriores propuestas "CSS 2D Transforms"[72],"CSS 3D

[71] http://www.w3.org/TR/css3-transforms/

[72] http://www.w3.org/TR/css3-2d-transforms/

Transforms"[73] y "SVG Transforms".

Para los conocedores de herramientas como Adobe Flash o Silverlight, podemos decir que la especificación de transformaciones se parece bastante, en cuanto al propósito, a sus equivalentes de estas herramientas, y permiten trasladar, girar y escalar elementos en espacios 2D y 3D sin necesidad de un complemento externo.

> **Nota**: Aunque en las versiones iniciales de IE10 era precisa la utilización de la extensión de navegador ***–ms***, no sucede así con las últimas versiones, que admiten el uso de las reglas tal y como están definidas en el estándar.

Tanto las transformaciones 2D como 3D se aplican a un elemento cualquiera utilizando la regla ***transform***, a la que se asignan como argumentos los parámetros requeridos separados por un espacio.

La lista de propiedades vinculadas con las transformaciones es la que vemos en la tabla siguiente:

Propiedad	Descripción
transform	Aplica una transformación 2D/3D a un elemento
transform-origin	Establece el punto de origen par alas transformaciones
transform-style	Indica cómo se disponen en un espacio 3D elementos anidados
perspective	Especifica la perspectiva para los elementos 3D
perspective-origin	Indica la posición inferior de elementos 3D

[73] http://www.w3.org/TR/css3-3d-transforms/

Tabla 8: Definiciones para las transformaciones

Propiedades relacionadas con las transformaciones

Y en la tabla adjunta, vemos una lista de algunos valores posibles para las transformaciones 2D.

Propiedad	Descripción
`rotate(<angle>)`	Rota un ángulo respecto a su origen tal y como se haya definido en la propiedad *transform-origin*
`rotateX(<angle>)`	Rota un ángulo en el eje de las X
`rotateY(<angle>)`	Rota un ángulo en el eje de las Y
`scale(<num×2>,)`	Escalado
`scaleX(<number>)`	Escalado en el eje de las X
`scaleY(<number>)`	Escalado en el eje de las Y
`skew(<angle×2>,)`	Deformación o Elongación para un ángulo
`skewX(<angle>)`	Deformación en el eje de las X
`skewY(<angle>)`	Deformación en el eje de las Y
`translate(<length×2>,)`	Desplazamiento (en el plano X/Y)
`translateX(<length>)`	Desplazamiento en el eje de las X
`translateY(<length>)`	Desplazamiento en el eje de las Y
`matrix(<various×6>,)`	Matriz de transformaciones 3x3 capaz de representar cualquiera de las anteriores individualmente o de forma combinada

Tabla 9: Propiedades vinculadas con las Transformaciones 2D

En cuanto al significado de los números, un valor positivo desplaza hacia la derecha o en el sentido de las agujas del reloj, mientras que uno negativo lo hace en sentido contrario.

Propiedad *transform*

Supongamos que tenemos un gráfico, que por razones de comparación repetimos dos veces (una a continuación de la otra), y aplicamos una transformación al segundo para observar la comparativa.

El marcado HTML podría ser, simplemente:

```
<img id="original" src="Imagenes/outlook8.png" />
<img id="ol" src="Imagenes/outlook8.png" />
```

Y para la transformación, definimos una regla para el segundo elemento, de forma que se vea en todos los navegadores y reduzca el tamaño del gráfico a un 75% de su valor original, al tiempo que se efectúa una rotación de 30 grados en sentido de las agujas del reloj:

```
#ol
{
    -moz-transform: scale(.75) rotate(30deg);
    -webkit-transform: scale(.75) rotate(30deg);
    -o-transform: scale(.75) rotate(30deg);
    transform: scale(.75) rotate(30deg);
}
```

Esto produce una salida como la de la figura 48, que se aprecia en todos los navegadores de forma prácticamente idéntica:

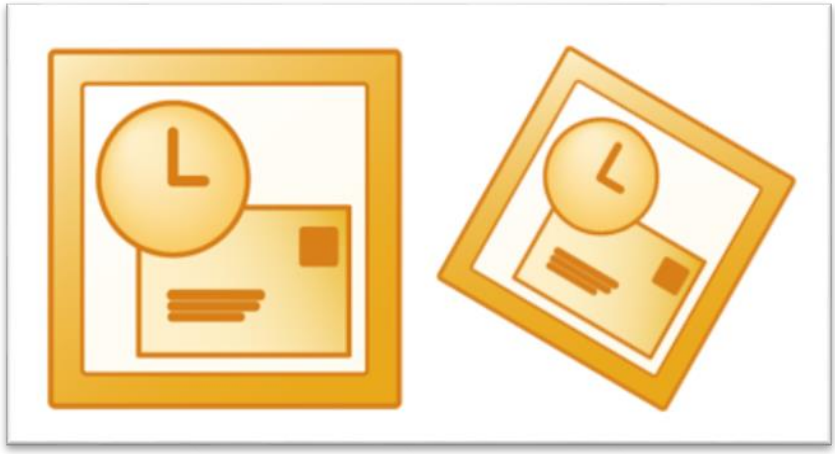


Fig. 48: Resultados de aplicar una transformación 2D

De la misma forma, podemos combinar los valores indicados en la Tabla de Transformaciones, indicando cada una de ellas separada por un espacio. En algunos casos, como el de las transformaciones de rotación en uno de los ejes, es preciso indicarle esta circunstancia al ***elemento contenedor***.

Como esto es aplicable a cualquier elemento, significa que podemos utilizar bloques de texto normales para dotar a este de ciertos efectos de perspectiva. Por ejemplo, podemos utilizar un bloque de texto como el de los ejemplos anteriores, y aplicar una de esas matrices de transformación 3D en la forma siguiente:

```
div
{
   transform: matrix3d(0.359127, -0.469472, 0.806613, 0,
0.190951, 0.882948, 0.428884, 0, -0.913545, 0, 0.406737,
0, 0, 0, 0, 1);
}
```

El resultado es que movemos el bloque entero girándolo en el eje de las Y, y obtenemos la proyección 2D resultante, como vemos en la figura siguiente:

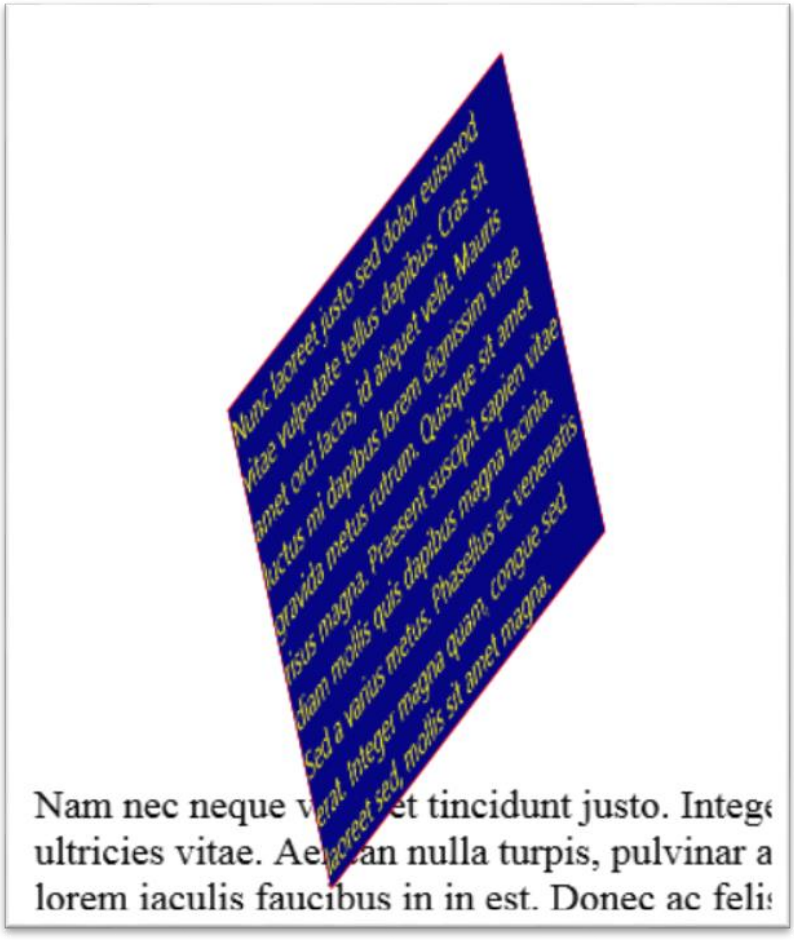


Fig. 49: Resultado de aplicar una transformación de rotación sobre el eje Y respecto a un bloque de texto

Donde hemos omitido el resto de valores de formato, donde se establece la anchura, el color de fondo y primer plano, etc.

También disponemos de propiedades que permiten indicar el punto de origen (***transform-origin***) o también un porcentaje respecto al original. Los valores posibles de esta propiedad son (a elegir uno o dos de los siguientes separados por un espacio):

- ***length***: Un número en coma flotante, seguido de un indicador absoluto de unidades (*cm*, *mm*, *in*, *pt* o *pc*) o un indicador de unidades relativas (*em*, *ex*, o *px*), que establece el origen de la transformación
- ***percentage***: Entero seguido de un %. El valor es un porcentaje de la longitud total de la caja (para el primer valor) o la altura del cuadro total (para el segundo valor, si se especifica).
- ***Left, center*** o ***right***: Se toman como primeros valores solamente.
- ***Top***, ***center*** o ***botton***: Se toman como segundos valores solamente.

Por ejemplo: `transform-origin: 50% 100%;`

Establece el punto de origen de la transformación a la mitad del primer valor predeterminado, y deja intacto el segundo.

Además, si combinamos estas propiedades y añadimos la propiedad ***perspective***, que habilita un efecto de alejamiento del objeto, podemos aplicar el siguiente código al mismo bloque anterior:

```
transform: perspective(500px) scaleZ(2) rotateX(45deg);
```

Obteniendo una salida como la de la figura 50:

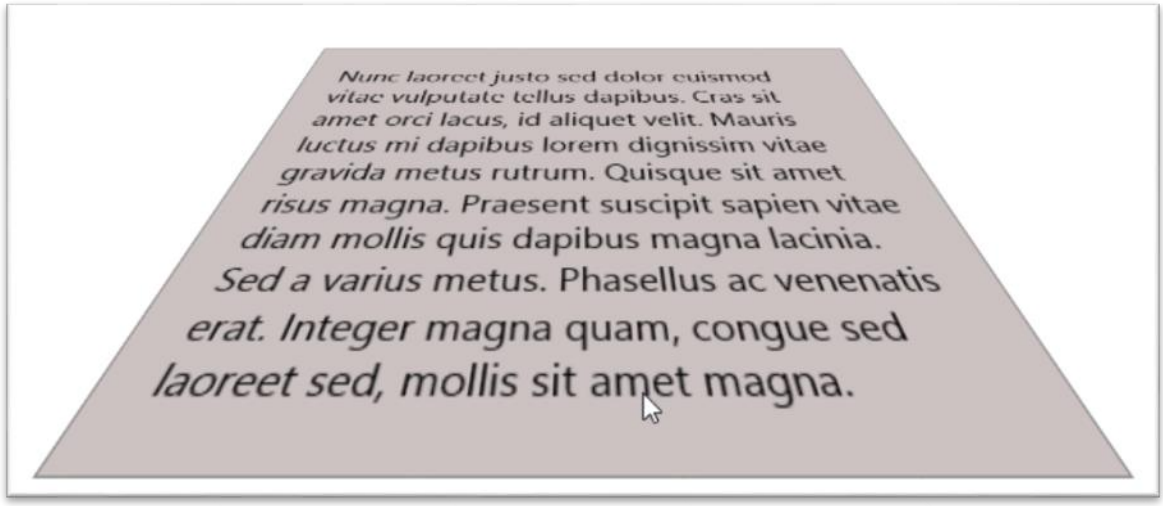


Fig. 50: Aplicación simultánea de efectos de escala, perspectiva y rotación.

Y nos basta con modificar el eje de rotación del bloque para obtener efectos radicalmente diferentes.

Por ejemplo, si cambiamos la rotación anterior de eje al de las Y:

```
transform: perspective(500px) rotateY(45deg);
```

El efecto final, como siempre sucede con este tipo de transformaciones, produce una sensación visual muy diferente, tal y como vemos en la figura 51:

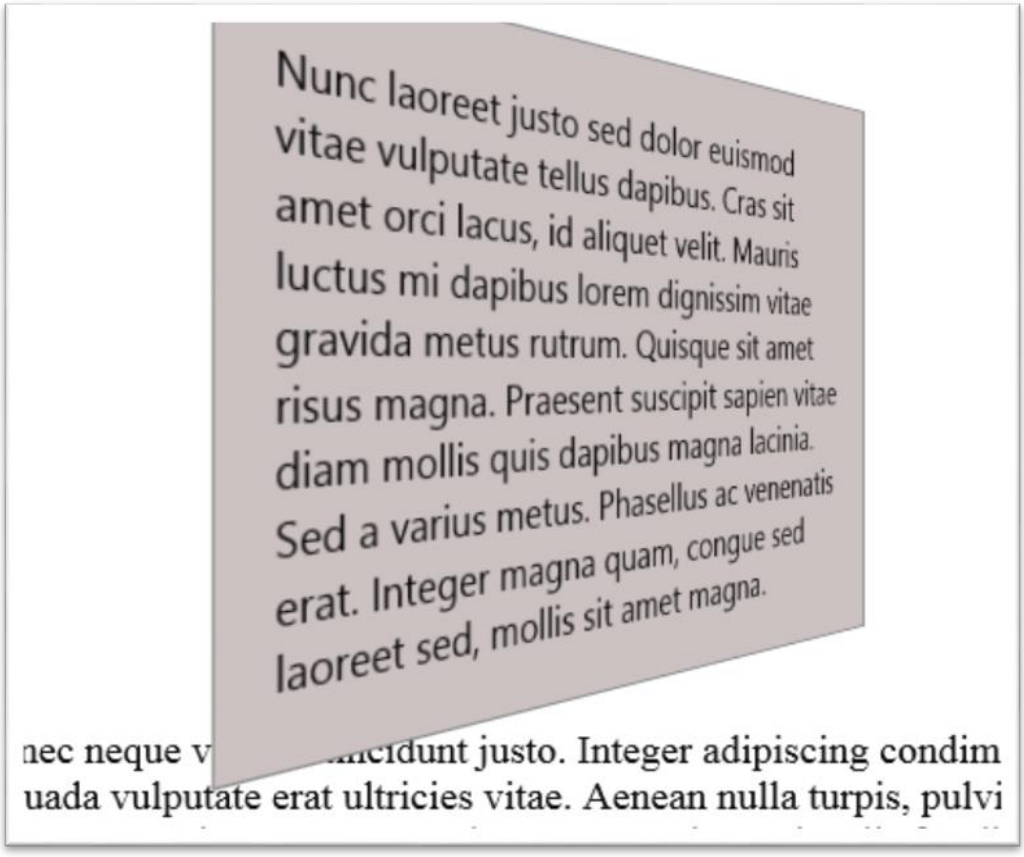


Fig. 51: resultado de cambiar el eje de transformación del código anterior

Si además utilizamos la propiedad ***transform-style***, podemos definir cómo se representan los elementos anidados en un espacio 3D. La documentación oficial del MSDN indica que "*si en un elemento se configura esta propiedad en plano, todos los elementos secundarios se representarán aplanados en el plano 2D del elemento. Si se rota el elemento alrededor de los ejes* **x** *o* **y**, *los elementos secundarios situados en posiciones* **z** *positivas o negativas aparecerán en el plano del elemento* **y** *no en el frente ni en la parte posterior*".

Finalmente, La propiedad ***backface-visibility*** indica si la cara posterior de un elemento transformado es visible cuando se orienta hacia el espectador. En un elemento no transformado, la cara anterior del elemento queda de frente. Sus valores posibles son ***visible*** (reverso visible), y ***hidden*** (reverso invisible).

Transiciones y Animaciones

Sin dinamismo las transformaciones son útiles pero carecerían de una buena parte de la fuerza que las hace ser tan populares en otros entornos como Silverlight o Flash. Con ese objetivo y para suministrar a las modernas interfaces de usuario de una experiencia diferenciadora, la W3C ha decidido crear estas dos especificaciones que se centran únicamente en elementos dinámicos.

Diferencia entre transiciones y animaciones

La principal diferencia entre una animación y una transición es que una transición sólo puede cambiar de un elemento a otro como resultado de una acción o cuando se convierte en un elemento activo, es decir, a través de una pseudo-clase. Sin embargo, las animaciones se pueden iniciar desde la carga de la página sin ninguna interacción del usuario, aunque puede mezclarse con dichas interacciones.

Ambas tienen sus propias ventajas, y son útiles en escenarios concretos. No obstante, si se requieren efectos complejos, tendremos que usar animaciones, que implican la idea de fotograma clave (***keyframe***),

característica de las animaciones.

Transiciones

Una transición, como lo define el documento oficial de la W3C, "*permite que los cambios de los valores en propiedades CSS se produzcan a lo largo de un período de tiempo predeterminado*".

El documento oficial en la actualidad es el "*CSS Transitions*"[74], cuyo último borrador de trabajo es de Abril/2012, en el momento de escribir estas líneas. En caso de duda, es importante su consulta, ya que contiene un apartado (*7.1 Properties from CSS*) que indica concretamente qué propiedades se pueden animar por esta técnica, ya que no lo son todas.

Tenemos, por lo tanto, dos factores totalmente nuevos: por un lado hay que manejar dos estados (el inicial y el final), y por otro, tenemos que utilizar la idea de duración, que define cómo se produce esa transición a lo largo del tiempo.

Propiedad *transition*

Para definir una transición se utilizan reglas ***transition*** y sus variantes, que permiten parámetros opcionales para indicar conjuntos de 3 o 4 valores que definan el comportamiento. El editor de código de Visual Studio 2012, nos muestra su formato:

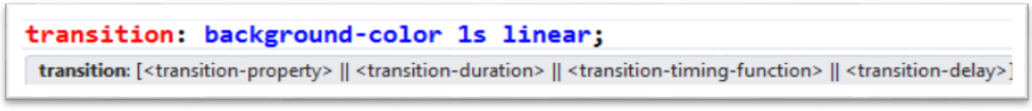


Fig. 52: definición de la propiedad transition en el editor de Visual Studio 2012

En uno de sus ejemplos iniciales, la W3C propone una situación simple, que recoge estos dos factores de forma sencilla:

```
li:hover
{
    background-color: yellow;
```

[74] http://www.w3.org/TR/css3-transitions/

```
    transition-duration: 2s;
}
```

Se trata de establecer una transición para los elementos **<*li*>**. La primera regla del código anterior establece el color de destino (a partir del inicial que tenga el elemento). La segunda, indica la duración del efecto. El resultado final será que, cuando el cursor pase por encima un elemento **<*li*>**, a lo largo de 2 segundos se producirá una transición entre un color y el otro. Cuando el cursor se retire, el color será devuelto a su estado original en un tiempo 0.

Como en este caso la transición es de tipo color, el sistema generara una serie de colores intermedios, por los que irá pasando el elemento hasta llegar a su estado final, como podemos ver en la figura 53:

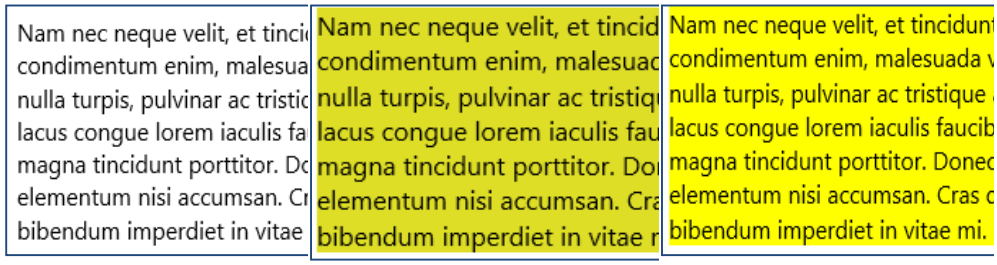


Fig. 53: Transición de estado entre dos colores. Las capturas muestran los estados inicial, intermedio y final.

También existe una forma sencilla de indicar la duración del segundo proceso (el de la vuelta al estado original). Para ello, podemos usar una definición estándar para el estado normal, junto a los parámetros de la transición: la propiedad de destino, un valor de duración y –eventualmente- un modo de hacerlo. Por ejemplo, el siguiente código:

```
li
{
    transition: background-color 1s linear;
    background-color:#c8bfbf;
}
```

Permite definir el color original, y también que cuando la transición concluya, se restaure ese color, que el proceso tiene que durar 1 segundo, y se haga utilizando el parámetro ***linear***, que se aplica a la ***transition-timing-function***, seleccionando una entre varias funciones predefinidas para realizar la transición opuesta.

transition-timing-function

Esta función, puede ser una línea recta (valor ***linear***), indicando que los valores que toma la función se modifican a saltos graduales, o puede ser una curva Bézier que, por ejemplo, indique que el proceso será lento al principio, rápido en su bloque central, y lento de nuevo al final. La siguiente imagen muestra una curva Bézier tradicional, que definiría un proceso de este tipo:

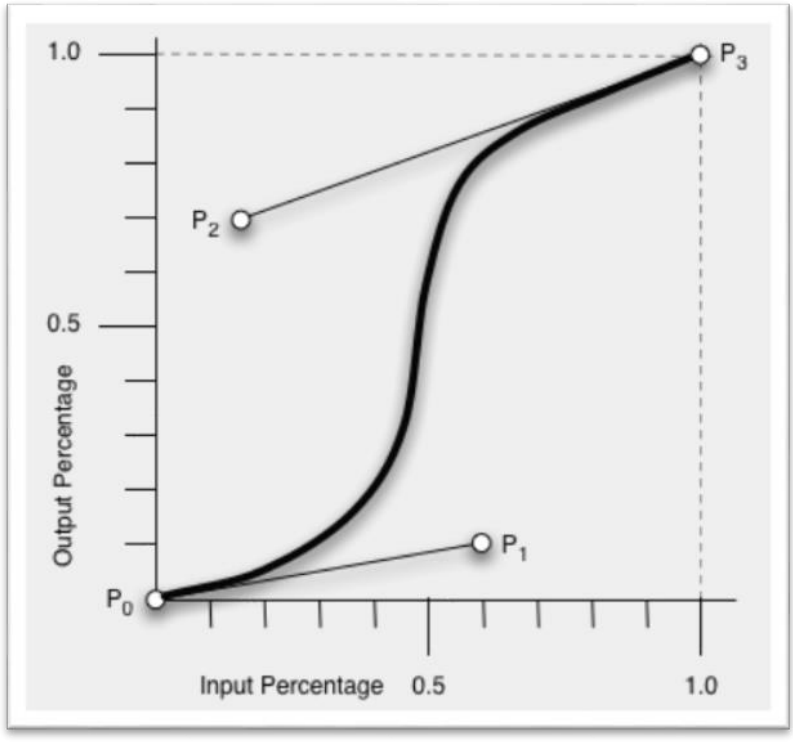


Fig. 54: timing-function definida mediante una curva Bézier a lo largo de un segundo de duración.

El estudio de este tipo de curvas se sale del objetivo de este libro, por lo que nos remitimos al documento original del estándar donde se encuentra una explicación detallada de sus características, en caso de que el lector necesite ayuda adicional en este punto.

Baste decir que el estándar define una serie de valores posibles para este parámetro, que son:

- ***ease***: equivalente a **cubic-bezier**(0.25, 0.1, 0.25, 1.0), y predeterminado. Se corresponde al comportamiento de la curva Bézier de la figura 43.
- **linear**: equivalente a **cubic-bezier** (0.0, 0.0, 1.0, 1.0).
- **ease-in:** equivalente a **cubic-bezier** (0.42, 0, 1.0, 1.0).
- **ease-out**: equivalente a **cubic-bezier** (0, 0, 0.58, 1.0).
- **ease-in-out**: equivalente a **cubic-bezier** (0.42, 0, 0.58, 1.0)
- **step-start**: equivalente a dos pasos (1, start).
- **step-end**: equivalente a dos pasos (1, end).
- **steps**: Indica una función que describe varios pasos, como la de arriba, y acepta dos parámetros. El primero es el número de intervalos de la función y debe ser un entero positivo. El segundo, que es opcional, puede valer bien '***start***' o '***end'***, e indica el punto donde ocurre el cambio de valores durante el intervalo. Si se omite el segundo, se asume "***end***".
- **cubic-bezier**: Especifica una curva Bézier de tipo cúbico. Los 4 valores indican los puntos P1 y P2 de la curva, y los valores del eje de las **x** deben de pertenecer al intervalo cerrado [0, 1] o la definición no será válida. Para propiedades que no sean ***opacity*** y ***color***, la función acepta coordenadas Y fuera del intervalo estándar comprendido entre 0 y 1. Esto permitirá crear efectos de transición "elásticos" o de "rebote".

Además de lo anterior, es posible indicar igualmente un parámetro denominado ***transtition-delay***, que permite establecer el tiempo que

debe transcurrir desde que la pseudo-clase se active hasta el inicio del efecto.

Las transiciones y la programación

Aparte de las habituales manipulaciones del DOM mediante JavaScript que podemos aplicar a este modelo, cuando una transición termina, se produce automáticamente el evento ***transitioned***, que puede ser capturado en elementos superiores (***event bubbling***) y también puede ser cancelado.

Este evento proporciona un par de parámetros de contexto: ***propertyName*** (la propiedad sobre la que se ha efectuado la transición), y ***elapsedTime***, que indica el tiempo invertido.

De cualquier forma, y, aunque el DOM lo hace posible, para el manejo de transformaciones y transiciones no es precisa la intervención de código JavaScript, con lo que se reducen, además, los tiempos de carga y ejecución.

Animaciones

Como indicábamos al principio de este apartado, la diferencia respecto a las animaciones es –en muchos casos- el grado de complejidad. Se trata de una especificación distinta (*CSS Animations*[75]), de fecha de Abril/2012, y que se encuentra en el estado "*Borrador de Trabajo*".

En este caso, de lo que se trata es de obtener más control sobre cómo el proceso de transición entre un estado inicial y el final tiene lugar. Para ello, se recurre a la idea de estado intermedio, cuyo concepto es definido mediante un objeto especial llamado *fotograma clave* o ***keyframe***.

Cada uno de estos objetos define exactamente un estado intermedio: el tiempo que se tarda en llegar a él, si se puede repetir el proceso mediante una iteración, si se puede invertir la flecha del tiempo (simulación "hacia atrás"), e incluso si deben efectuar pausas y reanudaciones.

[75] http://www.w3.org/TR/css3-animations/

Desde código, lo primero es ubicarse en el selector que corresponda al elemento de destino, y definir un estado final para la animación. Por ejemplo, para animar un párrafo concreto (que ya hemos formateado previamente en ancho, alto y color de fondo), podemos escribir lo siguiente:

```
p:hover
{
    /*Animación*/
    animation-name: demo;
    animation-duration: 3s;
    animation-timing-function: ease-out;
    animation-delay: -1s;
    animation-iteration-count: 2;
    animation-direction: normal;
}
```

Donde, vemos que la primera propiedad significativa es el nombre de la animación (necesario para vincular los fotogramas clave), y le siguen un conjunto de reglas derivadas de ***animation-***, para establecer la duración total del proceso, la función utilizada, el retraso en la puesta en marcha, el número de iteraciones, y la dirección.

También podemos indicar todo esto en una sola instrucción, (separando los valores por espacios) y recordando que las unidades de medida deben de expresarse indicando la unidad a utilizar:

```
animation: myAnimation 3s ease-out -1s 2 normal;
```

El siguiente paso es establecer los puntos intermedios, cosa que se consigue con la directiva ***@keyframe***. Esta directiva adopta una estructura formal de este tipo:

from {...} [Punto intermedio 1 {...}, Punto intermedio 2 {...}, ...] to {...}

Donde se indica el punto inicial (***from***, equivalente a 0%) y el último (***to***, equivalente a 100%) para establecer los puntos de inicio y fin, y se

permite un número indeterminado de valores intermedios. Cada punto intermedio se indica mediante un valor de porcentaje sobre el total, de forma que podríamos tener un código como el siguiente:

```
@keyframes demo
{
    from
    {
        animation-timing-function: ease;
    }

    50%
    {
        background-color: purple;
        animation-timing-function: ease-in;
        transform: translate(20px,30px);
    }

    to
    {
        background-color: blue;
    }
}
```

Lo que significa: Asignamos una colección de 3 ***keyframes*** a la animación llamada ***demo***, consistentes en un elemento inicial que empieza con una función de animación tipo ***ease***. Al 50% del tiempo transcurrido se deberá de haber cambiado el color a ***purple,*** y se trasladará el objeto 20 píxeles en el eje de las X y 30 píxeles en el de las Y. Además, desde ese punto al siguiente ***keyframe*** (que, en este caso, coincide con el del final), el proceso utilizará una función ***ease-in***. Por último, cuando la animación termina, el color de fondo será azul.

> Nota: Hay que tener en cuenta que probablemente, tendrá que añadir las instrucciones que comprenden el resto de navegadores, incluyendo, al menos, Firefox y Chrome.

Con este escenario, ni siquiera necesitamos lanzar el proceso mediante un evento, ya que hemos utilizado la pseudo-clase ***hover***, por lo que se pondrá en marcha cada vez que pasemos el cursor por encima del elemento.

El resultado es el previsible, dado que –salvo por los fotogramas clave- la situación es muy similar a la que teníamos mediante el uso de transiciones. En la figura adjunta mostramos los estados inicial, intermedio y final de la animación:

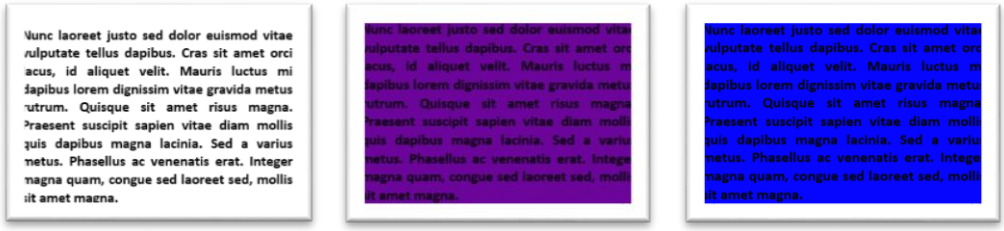


Fig 55: Tres capturas en ejecución del código anterior, mostrando los 3 estados.

Finalmente, si deseamos lanzarlo mediante código JavaScript, es posible que el procedimiento no sea muy intuitivo a primera vista, ya que la animación cambia al cambiar de estado.

La invocación por código y las pseudo-clases

Afortunadamente, la posibilidad de establecer dinámicamente ese cambio de estado permite utilizar pseudo-clases como ***:target.***

De esta forma podemos definir un estado inicial simple y uno final que implique una animación, a cuyas propiedades habremos añadidos la regla ***animation-play-state***, cuyos valores pueden ser: ***running*** (el predeterminado, que provoca la ejecución de la animación) y ***paused*** que

detiene la ejecución en el punto donde se encontrase. Más adelante, podemos reanudarla con otra asignación a ***running.***

Otra propiedad asociada con las animaciones, ***animation-fill-mode*** admite 3 valores específicos (además de los globales): ***forwards, backwards*** y ***both.*** Y permite definir esa especie de "flecha del tiempo" que indicábamos antes.

Podemos probar esto escribiendo las siguientes definiciones CSS para un elemento ***<p>:***

```
p
{
    /*formato*/
    width: 300px;
    height: 200px;
}
p.target
{
    animation: demo 3s ease-out -1s 2 normal forwards;
    animation-play-state:running;
}
```

Posteriormente, solo tenemos que asignar el evento ***click*** de un botón a una llamada a JavaScript que ponga en marcha el cambio de clase. Si disponemos de jQuery, esto resulta tan sencillo como añadir el siguiente código fuente:

```
<!-- Asumimos que existe este botón con su identificador
y que el párrafo sobre el que actuamos se llama "Paranim"
-->

<p id="Paranim">Nunc laoreet justo sed dolor euismod
vitae vulputate tellus dapibus. Cras sit amet orci lacus,
id aliquet velit. Mauris luctus mi dapibus lorem
dignissim vitae gravida metus rutrum. Quisque sit amet
risus magna.</p>
```

```
<input type="button" value="Lanzar Animación" id="boton"
/>
<script type="text/javascript">
    $(function () {
        $("#boton").click(function () {
            $("#Paranim").toggleClass("target");
        });
    });

</script>
```

De esa forma, podremos llamar a la animación cada vez que se pulse el botón. De hecho, esta operativa es perfectamente válida también para las transiciones que vimos en el apartado anterior.

Por lo demás, las animaciones sí se reflejan en el DOM, que recoge hasta 3 posibles eventos que pueden producirse durante el proceso de lanzamiento y ejecución de una animación:

- **animationStart**: tiene lugar cuando se lanza una animación por cualquier medio.
- **animationIteraction**: que se produce cuando concluye cada una de las posibles iteraciones de una animación
- **animationEnd**: que se lanza cuando termina la animación

Todos ellos suministran información del nombre del evento o del tiempo transcurrido.

Alternativas para navegadores antiguos

Llegados a este punto, es tiempo de hablar de las situaciones excepcionales. Y son muchas, especialmente cuando hablamos de soporte "hacia atrás". Durante el período IE5-IE7 el soporte de CSS de Internet Explorer contenía algunas variantes de tipo propietario, lo que

hacía difícil programar con las características de hoy para que esas versiones del explorador interpreten correctamente las páginas que usan características avanzadas o –incluso- que hagan un "fallback" correcto.

Para corregirlo, desde el principio se han utilizado un conjunto de reglas especiales que solo son entendidas por esas versiones del navegador, llamadas **CSS Hacks**.

NO siguen el patrón actual de prefijos propietarios que hemos visto aquí, sino que se basan en ciertos caracteres que anteceden a la descripción de una regla, que los demás navegadores ignoran, pero que algunas versiones de IE, interpretan.

El sitio web http://caniuse.com, que mencionábamos en el primer capítulo, mantiene un estudio actualizado del nivel de soporte de todas las características del estándar en los diferentes navegadores, tanto en modelos de sobremesa como en los móviles. Se puede analizar el nivel de soporte de forma exhaustiva (cada característica individual tiene su apartado), y lo que es mejor, se puede obtener información de los distintos "***fallback***" existentes para una característica concreta.

Vamos a comentar aquí los más utilizados y populares:

Comentarios con barra invertida

- Este *hack* explota un bug en Internet Explorer para Mac relacionado con un comentario de análisis. Un comentario que termina en \ * / **no** está bien cerrado en IE Mac, por lo que las normas que deben ser ignoradas en IE Mac puede ser colocado después de un comentario. Otro comentario es necesario después de la regla para cerrar el comentario para IE Mac [3].

```
/ * No haga caso de la siguiente regla de IE mac \ * /
selector {... estilos ... }
/ * Dejar de ignorar en IE mac * /
```

Hack del modelo de caja

- Se utiliza para evitar que Internet Explorer genere errores del modelo de caja, y proporciona un conjunto diferente de propiedades a Internet Explorer y otros navegadores.
- A partir de la versión 6, IE ha corregido el bug modelo de caja en los documentos que incluyen DTD (DOCTYPE).

```
#elem {
  anchura: [IE ancho];
  voice-family: "\"} \ "";
  voice-family: inherit;
  anchura: [ancho en otros navegadores];
}
html> body #elem {
  anchura: [ancho en otros navegadores];
}
```

- La primera declaración **voice-family** se termina en la cadena "**}**", pero un error del analizador IE lo interpretará como una cadena con una barra invertida seguida de una llave de cierre para el extremo de la regla.
- Se elige **voice-family** porque no afectará a la representación en pantalla de la hoja de estilo. La segunda regla utiliza el truco **html > body** para navegadores como Opera 5 que tienen el error de análisis, pero no tienen el bug del modelo de caja (y, además, soportan el selector de hijo.

Hack del Guion Bajo

- Las versiones 6 (y por debajo) de Internet Explorer reconocen las propiedades con este prefijo (después de descartarlo). Los otros navegadores ignoran estas propiedades como no válidas. Por lo

tanto, una propiedad que está precedida por un guion bajo se aplica exclusivamente en Internet Explorer 6 y anteriores.

```
#Elem {
  anchura: [Ancho Modelo W3C];
  _width: [Modelo BorderBox en IE6];
}
```

- Este hack usa CSS no válido y hay directivas CSS válidos para lograr un resultado similar. Por otra parte, este truco no cambia la especificidad de un selector haciendo más fácil el mantenimiento de un archivo CSS.

Hack del asterisco (*)

- Las versiones 7 y siguientes de Internet Explorer reconocen las propiedades que están precedidos por caracteres no alfanuméricos excepto un guion o un guion (después de descartar el prefijo).
- Todos los otros navegadores ignoran estas propiedades no válidas. Por lo tanto, una propiedad que está precedida por un carácter no alfanumérico que no sea un guion bajo o un guion medio, como un asterisco, se aplican exclusivamente en Internet Explorer 7 y anteriores.

```
#elem {
  width: [Modelo W3C];
  * width: [Modelo BorderBox ];
}
```

- Este *hack* usa CSS válido y hay directivas CSS válidas para lograr un resultado similar.

Hack de HTML

- El elemento ***html*** es el elemento raíz del W3C DOM estándar, pero las versiones de Internet Explorer 4 a 6 incluyen un elemento primario misterioso. El resto de navegadores ignorarán el selector *** html**, mientras **IE4-6** lo procesará normalmente.
- Por ejemplo, esta regla especifica el tamaño del texto en Internet Explorer 4-6, pero no en los otros navegadores.

```
* Html p {font-size: 5em;}
```

- Este *hack* usa CSS válido por completo.

Hack del asterisco a partir de IE7 (*+)

- Aunque Internet Explorer 7 ya no reconoce el clásico truco del asterisco +HTML, se ha introducido un truco similar con nuevos selectores para IE7:

```
*: First-child + html p {font-size: 5em;}
 ó...
* Html + p {font-size: 5em;}
```

- Este código se aplicará en Internet Explorer 7, pero no en cualquier otro navegador. Este truco sólo funciona en IE7 modo estándar, no funciona en el modo de compatibilidad. Es compatible con la Vista de compatibilidad de Internet Explorer 8 (IE7 modo estándar), pero no en el modo estándar de IE8. Al igual que el truco *HTML, se utiliza CSS válido.

Hack del selector de descendientes

- Internet Explorer 6 y versiones anteriores no son compatibles con el "selector de descendientes" (**>**), permitiendo indicar normas para el resto de navegadores. Por ejemplo, esta regla vuelve azul un párrafo de texto en Firefox, pero no en IE antes de la versión 7.

```
html> body p {color: blue;}
```

- Aunque IE7 añade soporte para el selector de descendientes, se ha encontrado una variación que permite excluir a Internet Explorer 7.
- Cuando un comentario vacío se produce inmediatamente después de selector de descendientes, IE7 continuará con la regla que sigue, como en las versiones anteriores de IE.

```
html >/**/ body p { color: blue; }
```

Hack de negación de pseudo-clase

- Internet Explorer 8 y anteriores no son compatibles con la pseudo-clase de negación CSS3 ***:Not()***. La versión 9 ya lo incluye.

```
.MiSelector {
  color: black/ * valor para IE 8 y anteriores/ *
}

html:not([ie8andbelow]) .MiSelector {
color: red; / * valor para Chrome, Safari, Opera, Firefox
y IE9 + * /
}
```

- La pseudo-clase de negación acepta cualquier selector simple y aplica las siguientes propiedades a todos los elementos que no coinciden.
- Tenga en cuenta que el selector ***ie8andbelow*** no tiene sentido.

- Una variación de este hack usa pseudo-clase **:root**, que es tampoco es reconocida por Internet Explorer 8 y anteriores.

Hack de body:empty

- **:empty** se supone que selecciona sólo los elementos que no incluyen ningún contenido. Sin embargo, Gecko 1.8.1 y por debajo (usado en Firefox 2.0.x y anteriores) selecciona incorrectamente **body:empty** incluso cuando el elemento **body** tiene contenido (que por lo general debería).
- Esto puede ser aprovechado para reglas CSS exclusivos para Firefox 2.0.x y otros navegadores que utilizan el mismo motor de *renderizado*.

```
/ * Hacer desaparecer elementos p en Firefox 2.0.x y
abajo * /
body:empty p {
  display: none;
}
```

- Este hack usa CSS válido.

Hack !important

- **!important**, supuestamente asigna una relevancia superior al resto a una regla determinada.
- Internet Explorer 7 y anteriores, aceptan prácticamente cualquier cadena en lugar de –***important***- mientras que el resto de navegadores lo ignora. Esto se puede usar para indicar valores a IE7

```
/ * Poner el texto en azul en IE7 y anteriores, y negro
en todos los demás * /
body {
  color: black;
  color: blue !ie;
}
```

- **IE6** y anteriores, también tienen otro problema con estas declaraciones cuando el mismo elemento tiene la misma declaración efectuada anteriormente (p.e.2 declaraciones de **color**), y la primera declaración tienen una marca **!important**
- Esto debiera suponer que la primera tuviese precedencia sobre la segunda, pero IE6 no lo reconoce, hacienda que sea la última (proceso en cascada), la que prevalezca.

```
/ * Poner el texto en azul * IE6 e inferiores /
body {
   color: black !important;
   color: blue;
}
```

Propiedades dinámicas

- Entre las versiones 5 y 7, Internet Explorer ha apoyado una sintaxis propia para la aplicación de las propiedades CSS que cambian de forma dinámica, a veces referida como **expresiones CSS**.
- Las propiedades dinámicas suelen combinarse con otros *hacks* para compensar las propiedades no compatibles en versiones anteriores de Internet Explorer.

```
div {
  min-height: 300px;

  /* simula min-height en IE6 */
  _height: expression(document.body.clientHeight < 300 ?
"300px”: "auto");
}
```

Comentarios condicionales

- Se trata de comentarios en el código HTML que son interpretados exclusivamente por las versiones IE5+ hasta IE10, que deja de soportarla cuando se encuentra en "Modo Estándares" (si lo soporta en otros modos de compatibilidad).
- Una forma habitual de utilizarlo ha sido la siguiente (para comprobar el tipo de navegador, por ejemplo):

```
<!--[if IE 6]>
<p>You are using Internet Explorer 6.</p>
<![endif]-->
<!--[if !IE]><!-->
<p>You are not using Internet Explorer.</p>
<!--<![endif]-->
```

- Para más datos sobre este tipo de comentarios, ver:
    - http://en.wikipedia.org/wiki/Conditional_comment

Hack recomendado para utilizar alternativas a otras versiones

- MS recomienda disponer de versiones personalizadas con todas las definiciones correspondientes al soporte que queremos adoptar y cargarlas condicionalmente según la necesidad, manteniendo la limpieza del código CSS:

```
<head>
  <title>Prueba</title>
  <link href="all_browsers.css" rel="stylesheet"
type="text/css">
  <!--[if IE]>
      <link href="ie_only.css" rel="stylesheet"
type="text/css">
  <![endif]-->
  <!--[if lt IE 7]>
```

```
      <link href="ie_6_and_below.css" rel="stylesheet"
type="text/css">                          <![endif]-->
  <!--[if !lt IE 7]> <![IGNORE[--><![IGNORE[]]>
      <link href="recent.css" rel="stylesheet"
type="text/css">
  <!--<![endif]-->
  <!--[if !IE]>-->
      <link href="not_ie.css" rel="stylesheet"
type="text/css">
  <!-- <![endif]-->
</head>
```

Consideraciones de rendimiento

Concluimos este apartado con algunos consejos relacionados con el rendimiento (prestaciones) de las páginas que utilizan CSS. Son recomendaciones recogidas de múltiples sitios y de las que existe un consenso general.

- Utilizar nombres de clases específicos
- Evitar referencias a ancestros lejanos (más de 4 o 5 niveles de profundidad)
- Evitar –en lo posible- los selectores genéricos
    - **.vista-calendario div {}**
- Recordar que los selectores "***sibling***" son costosos, por la forma en que el motor los tiene que interpretar.
- El uso de "***box-shadows***" es costoso en contextos donde el contenido cambia constantemente.
- El posicionamiento absoluto (donde tenga sentido) está mucho más optimizado en todos los navegadores.
- Evitar las cargas de definiciones de CSS (ficheros externos, por ejemplo), durante las animaciones.

  - Podrían forzar una nueva "*renderización*", al tiempo que la animación está en movimiento.
  - Si realizamos alguna prueba con navegadores que soportan animaciones en 3D, es preferible cachear los contenidos y no alternar vistas 2D con 3D.

- Además, el cambio en la propiedad ***z-index*** podría comprometer todo el proceso (algún navegador "casca" al hacer esto).
- En todo lo que tiene que ver con animaciones, es preferible utilizar el API ***requestAnimationFrame,*** muy optimizado para ser utilizado en estos contextos, en lugar de sus equivalentes JavaScript anteriores.

Conclusión

Hasta aquí, todo lo relacionado con CSS y, concretamente, con las novedades que presenta CSS3 que no son pocas, y van en aumento a medida que progresa la especificación y aumenta paralelamente el soporte de los navegadores.

A partir de aquí vamos a centrarnos en la funcionalidad, y por tanto, en el lenguaje JavaScript y las API que han sido diseñadas para él.

Referencias

- "*CSS3. Visual QuickStart Guide*", **Jason Crawford Teague**, Ed. Peachpit Press, 2011
- "*Stunning CSS3*", **Zoe Mickley Gillenwater**, Ed. New Riders, 2011
- "*Pro HTML5 and CSS3 Design Patterns*", **Michael Bowers**, **Dionysios Synodinos** y **Victor Sumner**, Ed. APress, 2011
- "*CSS3 for Web Designers*", **Dan Cederholm**, Ed. A Book Apart, 2011
- "*Guía de Windows Internet Explorer 10 Release Preview para desarrolladores*", Microsoft, http://msdn.microsoft.com/library/es-

ES/hh673549.aspx

- "*CSS 3 Tutorial*", en el sitio http://www.w3schools.com/css3/default.asp
- *"Everything you Know about Clearfix is Wrong"*: *http://www.tjkdesign.com/articles/clearfix_block-formatting-context_and_hasLayout.asp*

Capítulo 05 | El lenguaje JavaScript 5

Tanto para los nuevos sitios Web, como para las aplicaciones basadas en HTML 5, el estándar prevé la utilización de un conjunto completo de recursos adicionales en forma de API programables (utilizando JavaScript), que aportan distinta funcionalidad, que va desde los datos de ubicación geográfica, hasta los "Web Sockets", e incluye mecanismos de almacenamiento de sesión y sistemas de caché locales, servicios más complejos de bases de datos basados en la propuesta *IndexDB*, notificaciones, aplicaciones "off-line", y unas cuantas más.

Pero, quizás, antes de plantearnos el funcionamiento de estas nuevas API, conviene revisar el propio lenguaje JavaScript, también ha sufrido una notable evolución.

JavaScript y ECMAScript

JavaScript comenzó en 1997, con una especificación inicial que se fue renovando casi anualmente, hasta llegar a ECMAScript 3ª Edición, que ha durado casi 10 años. En diciembre de 2009, apareció esta versión, y, actualmente, se trabaja en la versión 6, como podemos ver en la figura adjunta:

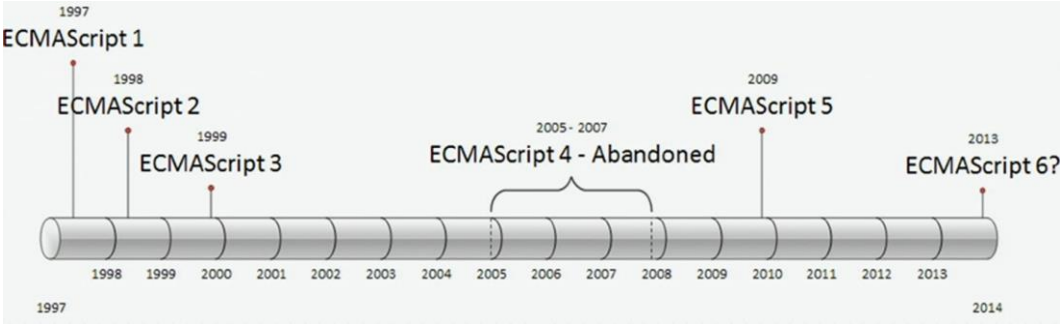


Fig. 1: Evolución de las versiones de JavaScript en el tiempo

JavaScript es el nombre popular de ECMAScript, el lenguaje estándar de la Web que implementa cada agente de usuario basándose en el documento de una entidad de normalización, ECMA International[76], cuya especificación 262 5ª Edición[77] (de Julio 2011), se encuentra ya completamente aprobado, y –en muy buena parte- implementado por los navegadores.

En el evento MIX.2011 de Microsoft, **Douglas Crockford**, uno de los principales responsables de la creación del lenguaje cuando se encontraba trabajando para Mozilla, hizo una presentación/resumen de lo más importante que debía considerarse en esta especificación, y cómo puede influir en esta Web de nueva generación.

[76] http://www.ecma-international.org/publications/standards/Ecma-262.htm

[77] http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf

Más recientemente, en Tech-Ed 2012 Europa, **Andrew Miadowicz**, *Program Manager* del equipo de desarrollo e integración de este lenguaje en Microsoft, presentaba cuáles son esas diferencias, cómo afectan al nuevo desarrollo y cómo se han implementado en Visual Studio 2012.

Vamos a resumir aquí lo más importante de lo que se dijo por ambos ponentes, así como en otras conferencias recientes sobre el tema, para dar al lector una idea del estado real actual del lenguaje, comenzando por el hecho de que –en palabras del propio Crockford en su charla del MIX11- IE10 es el primer navegador en cumplir totalmente con esta nueva especificación.

Pero comenzamos por un pequeño recordatorio de este controvertido lenguaje y algunas de sus "peculiaridades", muchas de las cuales fueron establecidas de esta forma "por diseño", en palabras de sus propios autores.

JavaScript: los puntos fuertes

Como todos los lenguajes interpretados (por oposición a compilados), JavaScript tiene puntos fuertes y otros más…extraños. Incluso los extraños, en su momento, tuvieron su función y su razón de ser, aunque ahora no veamos su propósito inicial.

Sintácticamente, es muy similar a Java, aunque ahí termina la similitud. Define un conjunto de objetos globales (accesibles desde cualquier punto), que ayudan a conseguir más fácilmente las tareas habituales, y en especial, aquellas para las que fue creado originalmente.

Así, tenemos objetos que se refieren a los tipos base, como ***Array, Boolean, Number, Object*** y ***String***. De forma que todo lo que se maneja son subtipos derivados de estos. Otros objetos complementan los anteriores o ayudan a su manipulación, como **Math** (operaciones aritméticas) o **RegExp** (expresiones regulares).

Algunas peculiaridades del lenguaje

En JS, cualquier variable puede convertirse de un tipo a otro, y el uso de

comillas simples o dobles es completamente intercambiable:

```
> var x = 1
undefined
> x = 2
2
> x = "hola"
'hola'
> x = 'adios'
'adios'
```

Igualmente podemos asignar valores booleanos o de *arrays* de forma directa:

```
> x = true
true
> x = false
false
> x = [1, 2, 'tres', "cuatro", false]
[ 1,
  2,
  'tres',
  'cuatro',
  false ]
```

Y la forma de acceder a los elementos es la esperada:

```
> x[1]
2
> x[2]
'tres'
> x[4]
false
```

Las cadenas también disponen de métodos sobrecargados muy útiles, que, en algunos casos- han pasado desapercibidas al programador ocasional. Eso sucede, por ejemplo con el método ***toString***(), que funciona como se espera habitualmente, pero admite un argumento que

permite convertir un número pasado como cadena en otro en binario, octal o hexadecimal, con solo indicarle la base a convertir:

```
> z = 1234
1234
> z.toString()
'1234'
> z.toString(2)
'10011010010'
> z.toString(16)
'4d2'
```

La declaración de un objeto sigue la notación JSON, que define un objeto como un conjunto de definiciones entre llaves de parejas *clave/valor* separadas por comas:

```
> o = { nombre: 'Monglorio', edad: 44, saldo: 200.5  }
{ nombre: 'Monglorio',
  edad: 44,
  saldo: 200.5 }
```

Otros objetos se crean como consecuencia de inicializar un objeto pasándole un valor a su constructor, como el objeto ***RegExp***:

```
> r = new RegExp('^\\d{5}$');
/^\d{5}$/
> r.test("83948")
true
```

Aquí definimos una expresión regular que permiten comprobar que el contenido puede convertirse en un dígito de 5 cifras. Esto también nos sirve para otros propósitos, ya que JS considera que cualquier asignación a una variable que comienza y termina por el signo **/** es una expresión regular. Así que podemos definir una expresión que compruebe un precio, por ejemplo usando la notación habitual de euros y decimales de la siguiente forma:

```
> r = /[\d+.?)=(\d)*]+/;
/[\d+.?)=(\d)*]+/
> r.exec('El precio es 99.33€')
[ '99.33',
  index: 13,
  input: 'El precio es 99.33€' ]
```

El método ***exec*** analiza el contenido, reconoce la cantidad de forma correcta, indica el índice donde empieza la cantidad, y hace "eco" de la cadena de entrada. Y lo mismo sucede si a una expresión de cadena le aplicamos el método ***match*** pasándole dicha expresión regular como argumento:

```
> 'El precio es 99.33€'.match(r)
[ '99.33',
  index: 13,
  input: 'El precio es 99.33€' ]
```

Pero uno de los usos más potentes de las expresiones regulares es como mecanismo de reemplazo dentro de una cadena. Por ejemplo, podemos escribir lo siguiente para reemplazar el precio indicado por otro en la misma expresión de antes:

```
> 'El precio es 99.33€'.replace(r, '55.44')
'El precio es 55.44€'
```

El manejo de fechas está soportado mediante el objeto Date, que también tiene algunas peculiaridades interesantes, como creación de fechas de cualquier tipo, o del tipo que corresponda a la fecha actual, o la posibilidad de realizar aritmética de fechas (sumando y restando cantidades a una fecha):

```
> f = new Date()
```

```
Mon Aug 26 2013 13:21:53 GMT+0200 (Hora de verano
romance)
> f = new Date()
Mon Aug 05 2013 13:22:24 GMT+0200 (Hora de verano
romance)
> f = new Date(2013, 5, 22)
Sat Jun 22 2013 00:00:00 GMT+0200 (Hora de verano
romance)
> f = f - 5
1371851999995  //Notación numérica interna de fechas.
> g = new Date(f)
Fri Jun 21 2013 23:59:59 GMT+0200 (Hora de verano
romance)
```

En cuanto a las estructuras de control, prácticamente se sigue la sintaxis de Java en bucles clásicos (***for/next***), bucles ***while***, bucles ***do...while***, y sentencias condicionales ***if-else*** y ***switch***, por lo que no vamos a repetirlo aquí, pero hay una estructura que funciona de forma un tanto peculiar y conviene no confundir con otros equivalentes (como **for-each**): es el bucle ***for-in***.

Este bucle está pensado en realidad para permitir recorrer las propiedades de un objeto cualquiera, por lo que tenemos la oportunidad de inspeccionar en tiempo de ejecución cuáles son las propiedades de un objeto generado por nosotros, por ejemplo. Si lo aplicamos al objeto anterior, obtendríamos:

```
> for (var clave in o) {
... console.log(clave + ': ' + o[clave])
... }
nombre: Monglorio
edad: 44
saldo: 200.5
```

Otra peculiaridad aparece cuando declaramos funciones, ya que el carácter dinámico de las variables permite crear mecanismos de propósito general. Por ejemplo, el operador más (**+**) está sobrecargado

tanto para números como para cadenas; esto quiere decir que podemos crear una misma función que nos permita sumar dos números o concatenar dos cadenas:

```
> function Añadir(x, y) { return x+y; }
undefined
> Añadir(4,5)
9
> Añadir('Buenos ' , 'Días')
'Buenos Días'
```

Otros aspectos propios del lenguaje

JS tiene muchas otras variantes peculiares. Especialmente si lo comparamos con lenguajes orientados a objetos, como C# o Java. Por ejemplo, el uso del operador de igualdad (**==**), el uso del objeto **Global** (al que pertenecen todas las variables declaradas fuera de una función) y el funcionamiento de ciertas operaciones en coma flotante, son de los más controvertidos, así como la creación de objetos.

JavaScript y el concepto de igualdad

Una de esas peculiaridades de JavaScript reside precisamente en los operadores. Y, en concreto en el operador de igualdad (**==**). El problema es la peculiar forma de entender el concepto de transitividad de este lenguaje (si a=b entonces b=a). Y a esto se unen las palabras reservadas para expresar situaciones excepcionales, muy distintas en su sintaxis, pero muy similares en significado, solo cambiando algún pequeño matiz: ***false, undefined, null*** y ***NaN***.

Exponemos en la siguiente tabla un conjunto de expresiones lógicas (operador de igualdad) y sus resultados, según JavaScript:

Expresión	Resultado
`' ' == '0'`	***false***
`0 == ' '`	***true***
`0 == '0'`	***true***
`false == 'false'`	***false***
`false == 0`	***true***
`false == undefined`	***false***
`false == null`	***false***
`null == undefined`	***true***
`" \t\r\n " == 0`	***true***

Tabla 1: Algunas expresiones con el signo de igualdad (==) en JavaScript

Todavía hay más peculiaridades en el funcionamiento de estos operadores así como de los valores en sí. Por ejemplo, vea las siguientes expresiones:

```
> true + true
2
> true + false
1
> true - true
0
> true * 3
3
> true * 3 == 3
True
```

De hecho, la expresión ***if(!x)*** devuelve ***true*** si el valor de **x** es uno de los siguientes: ***undefined***, **null**, **0**, **''** ó ***false.***

La razón de estos resultados, puede hacernos pensar ya que –a veces- los razonamientos son un tanto sutiles y requieren una comprensión del lenguaje. Afortunadamente, existe un operador alternativo, el de igualdad estricta (el triple signo igual **===**) que si lo aplicáramos a los casos anteriores, nos devolvería ***false*** en todos los casos. La recomendación es utilizar éste siempre que haya problemas de posibles ambigüedades.

Otro aspecto peculiar es de la reasignación de valores a los métodos de cualquier objeto, ¡incluyendo algunos de los propios de JavaScript! Eso significa que podemos reasignar el funcionamiento de un método teóricamente cerrado como ***Math.random()***:

```
> Math.random()
0.4056333117187023
> Math.random()
0.6825279118493199
> Math.random = function() { return 9; }
[Function]
> Math.random()
9
```

Esto es debido a que no existe el concepto de clase sellada o inmutable, al estilo de lo que vemos en C#, por ejemplo.

Así que, quedémonos con el concepto de objeto como un identificador que podemos definir con la sintaxis comentada y cuyas propiedades (atributos) pueden tener valores de cualquier tipo, incluyendo una función asignada a ellos. Sin embargo, hay un detalle que debemos tener en cuenta: cualquier función definida de esta forma (o de otra de las posibles) no es llamada si no utilizamos la sintaxis especial de llamada que supone añadir al nombre de la función el doble paréntesis final:

```
> var persona = new Object();
undefined
> persona.nombre = "Filapiano";
'Filapiano'
```

```
> persona.saludar = function() {
console.log(this.nombre); }
[Function]
> persona.saludar
[Function]
> persona.saludar()
Filapiano
```

Donde la palabra reservada ***this*** siempre hace referencia al objeto al que se está haciendo referencia (el objeto ***persona***, aquí). De todas formas, volveremos más adelante sobre este término.

Además, dado que la asignación es siempre dinámica, podemos reasignar más de una variable al mismo objeto:

```
> var personaje = persona
undefined
> personaje.saludar()
Filapiano
```

Por la misma razón, un miembro de un objeto, puede ser otro objeto:

```
> personaje.dirección = { domicilio: 'C/Falla', CPostal :
47007 }
{ domicilio: 'C/Falla', CPostal: 47007 }
> personaje.dirección
{ domicilio: 'C/Falla', CPostal: 47007 }
> personaje.dirección.CPostal
47007
```

Y, así, podemos continuar el proceso de anidación sin más límite que lo razonable, realmente.

Otra forma habitual de creación de objetos es mediante lo que se denomina literales de función (***function literals***). La definición de la función es asignada a una variable y puede posteriormente ser "invocada" utilizando la sintaxis de llamada (el doble paréntesis, con o sin

argumentos) a que aludíamos antes:

```
> var f = function(param) { console.log('Valor pasado: '
+ param); }
> f
[Function]
> f()
Valor pasado: undefined
> f('datos')
Valor pasado: datos
```

También es posible mezclar la sintaxis de creación de un literal de función utilizando la palabra reservada ***this*** para indicar que ese literal contiene, a su vez, otra definición de función que podemos llamar desde fuera:

```
> var fun = function(parametro) {
      var saludo = 'Hola, soy ' + parametro;
      this.hola = function() { console.log(saludo); }
  }
> var m = new fun('Emerenciano')
> var t = new fun('Tesifonte')
> m.hola()
Hola, soy Emerenciano
> t.hola()
Hola, soy Tesifonte
```

Adviértase que la definición de ***saludo*** es privada al literal de función, por lo que, en este caso, no es posible modificarlo desde fuera. Esto es, si tratamos de reasignar ese valor, el proceso no funciona, con lo que tenemos un comportamiento que recuerda ligeramente el comportamiento de las clases en lenguajes compilados.

Además, una literal de función puede usarse como argumento de otra función, por lo que se evaluarán en dinámicamente en tiempo de ejecución, por lo que es posible escribir lo siguiente (basado en las definiciones del código anterior):

```
> var h = new fun('Hector')
> h.hola()
Hola, soy Hector
> function SaludoYFecha(saludo) {
..... console.log(saludo.hola() + ' estamos a ' + new
Date());
..... }
> SaludoYFecha(h)
Hola, soy Hector
estamos a Tue Aug 27 2013 12:12:18 GMT+0200 (Hora de
verano romance)
```

Y otra de las peculiaridades de las funciones que no encontramos habitualmente en los lenguajes orientados a objetos es que una función puede contener otra función, o puede devolver otra función. Por ejemplo:

```
var NumeroACadena = function () {
    var numeros = ['cero', 'uno', 'dos', 'tres',
'cuatro'];
    return function (n) {
        return numeros[n];
    };
}();
> console.log(NumeroACadena(4));
cuatro
```

La palabra reservada *this*

Una de las partes más oscuras del funcionamiento de JavaScript recae sobre el comportamiento de la palabra reservada ***this***, y el problema es que su valor es reasignado dinámicamente.

Cuando se llama a una función, ésta recibe implícitamente un argumento ***this.*** Muchas funciones no lo utilizan en absoluto. Pero imaginemos una situación como esta:

```
var objeto = {
    dato: 5,
    sumar: function (x, y) { return x + y + this.dato; }
```

```
}
objeto.sumar(1, 2) === 8;
```

Si llamo al objeto en la forma usual no hay problema, pero, como siempre se puede asignar una función a una variable podríamos hacer lo siguiente (con el resultado que se indica a continuación).

```
var f = objeto.sumar;
f(2, 3) === NaN;
```

¿Cómo es posible que se obtenga ***NaN*** si en la instrucción anterior funcionaba correctamente? La razón es que en el primer bloque de código, ***this*** hace referencia a **objeto**, pero en el segundo, ***this*** se refiere al objeto global, que no dispone de una propiedad ***dato***. Consecuentemente, al intentar realizar la suma, ***dato*** tiene el valor ***undefined***, que, al sumarse con un número, devuelve ***NaN.***

Estructuras de orientación a objetos en JavaScript

Como hemos visto, el lenguaje adolece de muchas de las cualidades que hacen a un lenguaje auténticamente orientado a objetos, si bien es cierto que posee muchas otras características similares. Una de las peculiaridades del lenguaje es que no existen clases. Los objetos no son más que "bolsas" de propiedades, pudiendo ser cada una de naturaleza distinta, incluyendo otros objetos, o funciones (ambos términos son muy similares, aquí).

También debe de haber quedado claro que las funciones pueden ser utilizadas como parámetros de otras funciones, y pueden ser asignadas a variables. De hecho, si el usuario no asigna una variable a una función, el entorno le asigna una variable con ese nombre, de forma automática. Debido a este tratamiento de las funciones, también es posible incrustar una función dentro de otra.

Por tanto, cuando creamos un objeto en JavaScript suceden algunas cosas curiosas "entre bambalinas". Por ejemplo si declaramos un objeto vacío, a continuación podemos llamar a su función ***valueOf***, u otra de las prexistentes en el objeto predefinido **Object**, ya que implícitamente,

hereda de ***Object*** su prototipo definido.

Prototipos

Esto sucede porque todos los objetos disponen de prototipos que ayudan a modelar su construcción. En el caso anterior, como el objeto vacío no tiene prototipo definido, buscará una propiedad con ese nombre dentro de **Object**, y la ejecutará. Si no la encontrase, seguiría buscando en la cadena de prototipos hasta producir un error.

Así que, podemos implementar algo similar a la herencia en JavaScript, mediante prototipos (***prototypes***), que hacen las veces de clases bien definidas. Si creamos un objeto vacío, este hereda un conjunto de elementos (funciones, por ejemplo) de su prototipo original (***Object.prototype***, al no indicarle otra cosa), y podemos usarlas directamente con ese objeto.

De hecho, si el ejemplo anterior lo hacemos en Visual Studio 2013, el editor de código nos lo indicará adecuadamente, como vemos en la imagen:

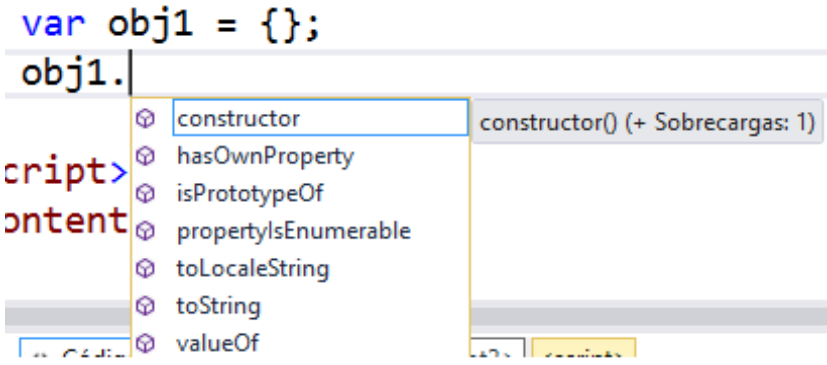


Fig. 2: métodos heredados del prototipo de Object al crear un objeto vacío en V. Studio 2013

Por tanto, podríamos llamar a la función ***valueOf*** en la segunda instrucción, y obtendríamos una salida indicando que se trata de [**object Object**], como se aprecia en la figura 3:

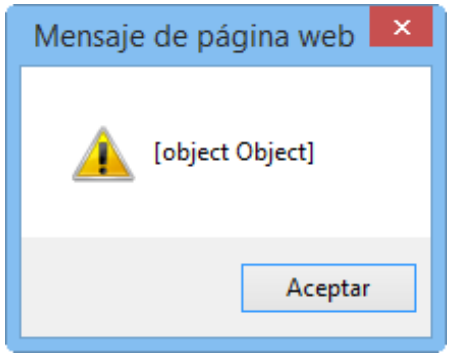


Fig. 3: Salida del código anterior.

En la práctica podemos referirnos al prototipo de un objeto mediante la palabra reservada ***prototype***, y realizar operaciones con él e incluso (si no está explícitamente prohibido) cambiar el prototipo.

En resumen, el motor de ejecución, cuando llamamos a una función de un objeto, recorrerá el objeto en busca de esa función, y, de no encontrarla, recorrerá todos los prototipos antecesores hasta encontrar una con ese nombre y la ejecutará, o devolverá un error del tipo " *El objeto no acepta la propiedad o el método –xxxx-*", si no la encuentra.

Funciones y el operador *new*

De forma que las funciones, y especialmente las que tienen estado (propiedades), pueden usarse de forma similar a lo que en otros lenguajes sería la definición de una clase. En parte debido al comportamiento dinámico de ***this***, es posible utilizar una definición de función junto al operador ***new*** para crear nuevas "*instancias*" de una clase "*definida*" por una función (y téngase en cuenta el entrecomillado de los términos *instancias* y *definida*)**.**

Por ejemplo, si tenemos la función *Persona* definida en apartados anteriores, podríamos crear nuevas *instancias* mediante la sintaxis:

```
function Persona(nombre, apellidos) {
    this.nombre = nombre;
    this.apellidos = apellidos;
```

```
}

var Luis = new Persona('Luis', 'Marcos');
var Irene = new Persona('Irene', 'García');

Luis.apellidos === 'Marcos';
….
Persona.prototype.nombreCompleto = function () {
    return this.nombre + " " + this.apellidos;
}

Irene.nombreCompleto() === "Irene García";
```

Como vemos en el código, después de crear la función podemos extenderla modificando su prototipo para añadir otra función adicional que nos devuelva el nombre completo. Al cambiar el prototipo, cualquier llamada a esa función por parte de cualquier objeto que herede del prototipo funcionará correctamente.

La razón es la peculiar "herencia" que existe en JavaScript. En el código que nos ocupa, cuando se utiliza ***new*** en la construcción, primero se crea un objeto vacío a partir de ***Object***. A continuación, establece su prototipo al que corresponda con ***Persona.prototype***. Finalmente ese objeto es pasado a la función ***Persona*** como un puntero ***this*** y la función queda inicializada.

> **Nota**: Por convención (aunque no es obligatorio), la notación de las funciones que ejercen de definiciones de objetos se escriben con la primera letra mayúscula.

No debemos confundir el prototipo con el constructor (***constructor***, en la sintaxis del lenguaje). Si consideramos el objeto ***Irene***, el esquema representativo de la herencia y sus relaciones de ancestros sería la siguiente:

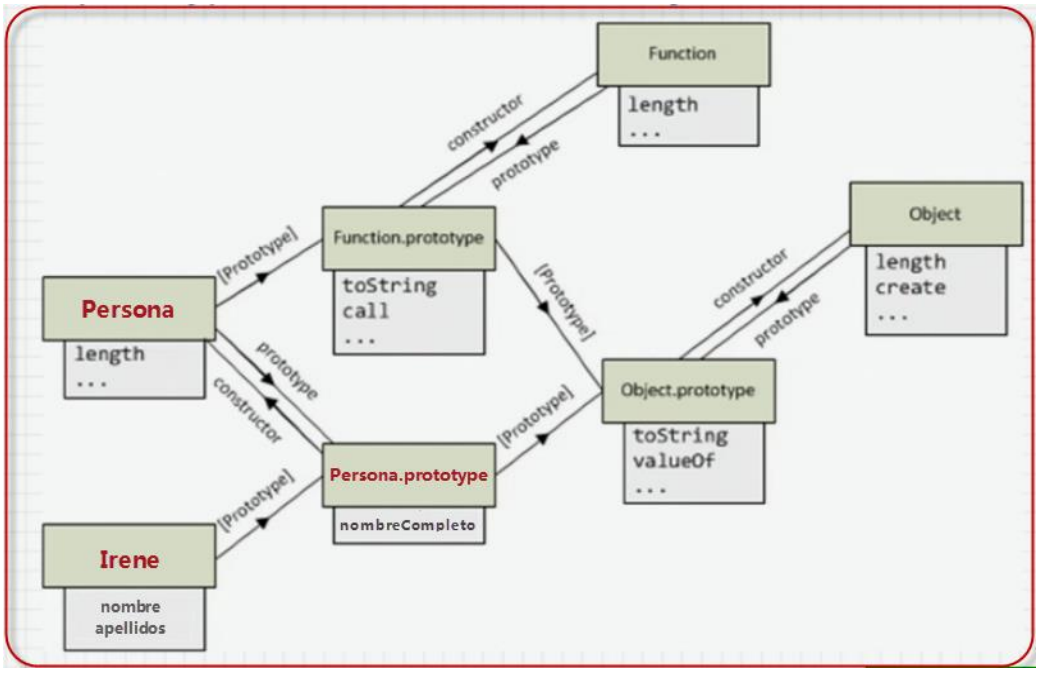


Fig. 4: Esquema de herencia del objeto Irene del código anterior

Como vemos en el esquema, el objeto *Irene* tiene unas definiciones propias (***nombre***, ***apellidos***), que compartiría cualquier objeto instanciado de esta forma. Pero también se ha definido una propiedad directamente en su prototipo (***Persona.prototype***), que cuenta con una propiedad extra (***nombreCompleto***). El prototipo, a su vez, dispone de un constructor ***Persona***, que dispone de una propiedad ***length***, por el hecho de heredar de ***Function***. Y así podemos seguir la cadena ascendente hasta el final.

El sistema parece un tanto lioso, pero así es como funciona. Por otra parte, las nuevas funciones de JavaScript 5, permiten progresar en la inspección de las cadenas de herencia hasta llegar al comienzo, sin importar si se trata de un elemento propio del lenguaje o de un elemento del DOM, ya que estos también heredan –al final de la cadena- de ***Object***.

Existen aún más detalles y modificaciones en el estándar JavaScript, así como otros que están siendo objeto de revisión y que, posiblemente, aparezcan en la versión 6, que ya se encuentra en fase de trabajo. Pero,

lo indicado aquí es lo más importante de cara al desarrollador. Remitimos a las referencias para más datos sobre el conjunto completo de novedades de ECMAScript 5.

Métodos y sobrescritura (*Overriding*)

Si bien no se trata de la característica que nosotros conocemos en otros lenguajes orientados a objetos, lo cierto es que esta forma declarativa del lenguaje permite sobrescribir métodos existentes, con la sencilla técnica de volverlos a definir en nuestra instancia del objeto.

Por ejemplo, para sobrescribir el método ***valueOf***, en nuestro código anterior, podemos indicar lo siguiente:

```
var obj2 = {
    valueOf: function () { return 3; },
    Propiedad1: 123
}
```

Nótese que debemos declarar la propiedad como una función, ya que ***valueOf*** es una función (si asignásemos un literal, se generaría un error de ejecución), por lo que el tipo definido el prototipo sí debe de respetarse.

El efecto es muy similar al de declarar una función con prefijo ***new*** en C#, ya que esta función va a ocultar cualquier implementación anterior.

Object.create() como mecanismo de inicialización de objetos

La nueva versión de JavaScript nos permite crear nuevos objetos a partir de cualquier otro definido por el contexto o por nosotros, mediante el método ***Object.create().***

Por ejemplo, podemos crear un nuevo objeto ***obj3*** basado en el anterior, y añadirle las propiedades que nos convenga:

```
var obj3 = Object.create(obj2);
obj3.Propiedad2 = "Valor";
```

El nuevo objeto contiene, entonces, la "herencia" dos prototipos: el suyo directo (***obj2***) y el del antecesor de éste (***Object***). Y su método ***valueOf***, tendrá el valor que reasignamos en su objeto padre (3: lo que devuelve la función).

Además, heredará por defecto los valores de su antecesor, de forma que su propiedad ***Propiedad1*** valdrá 123, como en el original, ya que –en realidad- apunta a aquél. De hecho, si creamos otro objeto a partir del mismo prototipo, también compartirá toda la funcionalidad y propiedades de su antecesor.

De esta forma, los prototipos en JavaScript cumplen una importante labor, tanto en la herencia como en la compartición de valores.

Estructuras tipo "*closure*"

Ya hemos visto que en JavaScript cualquier variable declarada fuera de una función pertenece al objeto ***Global***. Esto plantea problemas de todo tipo, especialmente, cuando nuestro código va a compartir el contexto de ejecución con otras librerías de terceros que podrían haber utilizado la misma notación.

Veamos el código siguiente:

```
var numeros = ['cero', 'uno', 'dos', 'tres', 'cuatro'];

var nombre_numero = function(n) {
    return numeros[n];
};

alert(nombre_numero(2));
```

Se declara la variable **numeros** en el objeto ***Global***, con las implicaciones indicadas antes. Una posible solución es incluir la definición de los números dentro de la función, pero eso obliga a inicializar el *array* **numeros** en cada llamada. Una mejor solución se basa en el concepto de ***closure.***

Según esta técnica, podríamos escribir una función que devolviese otra función, y que al añadirle el símbolo de llamada **()** fuerza –además- la ejecución de ésta.

El código tendría un aspecto como el siguiente:

```
//La misma solución con "closures"
var nombre_numero_c = function() {
    var numeros = ['cero', 'uno', 'dos', 'tres',
'cuatro'];
    return function (n) {
        return numeros[n];
    };
}();

alert(nombre_numero_c(2)); -> 'dos'
```

En la llamada a ***alert*** el argumento que se pasa a la función es "propagado" a la sentencia ***return*** y recogido por la variable ***n*** que utilizamos para el valor de retorno.

Además, el doble paréntesis del final de la definición de la función crea el objeto definido, pero NO lo inicializa cada vez que se usa, sino solo al cargarse la página (en la interpretación y asignación inicial). Con esta solución, disponemos del grado adecuado de encapsulación (evitando la inicialización repetida) y la funcionalidad es idéntica. Es una buena solución, muy utilizada en aplicaciones Web.

Otra estructura vinculada con esta idea y que se usa mucho en la actualidad consiste en "envolver" completamente entre paréntesis la definición de una función normal, y añadirle la sintaxis final de llamada, con lo que la función es invocada la primera vez que el ***Parser*** del navegador interpreta el código.

```
> (function f1(a, b) {
.....             console.log(a + b);
.....         })(1, 1);
2
> console.log(f1(3,4))
```

```
ReferenceError: f1 is not defined
```

Donde vemos que, al encerrar la definición de la función ***f1*** entre paréntesis, la estamos haciendo local y no global, y al incluir el doble paréntesis de llamada final con dos argumentos, forzamos a que se llame a la función y se le pasen los argumentos como parámetros, por lo que automáticamente nos presenta la respuesta.

Pero, por otro lado, si –a continuación- intentamos utilizarla como si su nombre fuera reconocible por el contexto, vemos que el entorno nos devuelve un error indicando que el identificador ***f1*** no está definido.

Con esto hemos revisado algunas de las particularidades más interesantes y necesarias del lenguaje en sus versiones clásicas. Vamos a revisar a continuación las novedades que presenta ECMAScript 5.

Lo nuevo en JavaScript 5

Esta propuesta que ha visto la luz con el nombre de JavaScript 5 incluye muchos cambios en el lenguaje, pero –a diferencia de la versión 4 que nunca vio la luz- no rompe la compatibilidad con los millones de páginas existentes: "*un cambio en el estándar, es siempre un acto de violencia*", afirmaba **Crockford** en su charla, aunque reconocía que "*a veces, con los estándares, es necesario un mal menor para conseguir un objetivo mayor*", en referencia a los cambios operativos de ECMAScript 5.

Entre estos cambios, cabe destacar las matrices tipadas, el nuevo modo ***strict***, nuevos métodos para la creación de objetos, propiedades de acceso, y varias novedades puntuales que actualizan, en general, la forma de utilizar el lenguaje. Vamos a revisar estas novedades con una breve descripción de cómo afectan a la programación.

Las Matrices Tipadas

La documentación de MSDN de Microsoft[78] nos ofrece una descripción del nuevo modelo de matrices tipadas, muy útil para el trabajo con objetos binarios en la red: La *API de matrices tipadas*[79] permite a las aplicaciones web utilizar diversos formatos de archivos binarios y manipular directamente el contenido, que ya sea compatible con *Windows Internet Explorer 10 Release Preview*.

Se pueden administrar datos binarios de orígenes como protocolos de red, formatos de archivo binario y búferes de gráficos sin procesar. También pueden usarse para administrar datos binarios en memoria con estructuras de bytes de definición conocida. Internet Explorer 10 lo soporta en JavaScript, en *XMLHttpRequest*, en la API de archivo y en la API de secuencias.

Las *matrices tipadas* ofrecen un modo de manejar el contenido de datos binarios sin procesar a través de una vista tipada específica. Por ejemplo, si queremos ver datos binarios sin procesar de un byte a la vez, podemos usar un Uint8Array. Si queremos leer los datos sin procesar como una matriz de números de coma flotante, podemos usar un Float32Array. Se admiten los siguientes tipos:

Tipo de matriz	Descripción y tamaño del elemento
Int8Array	Entero con signo de 8 bits
Uint8Array	Entero sin signo de 8 bits
Int16Array	Entero con signo de 16 bits
Uint16Array	Entero sin signo de 16 bits
Int23Array	Entero con signo de 32 bits
Uint32Array	Entero sin signo de 32 bits

[78] http://msdn.microsoft.com/es-es/library/ie/hh869304(v=vs.85).aspx

[79] http://go.microsoft.com/fwlink/p/?LinkID=245657

Float32Array	Número en coma flotante IEEE754 de 32 bits
Float64Array	Número en coma flotante IEEE754 de 64 bits

Tabla 2: Tipos de datos admitidos para las matrices tipadas

Existe una demo de matrices tipadas en acción en la dirección http://go.microsoft.com/fwlink/p/?LinkID=245671.

Para su funcionamiento, se utilizan dos objetos complementarios que ofrece esta API: ***ArrayBuffer*** y ***DataView.*** El primero es una referencia a los datos binarios sin procesar, pero no ofrece ninguna manera directa para interactuar con los datos. La creación de una vista ***TypedArray*** del ***ArrayBuffer*** proporciona acceso de lectura y escritura al contenido binario.

El objeto ***DataView***, por su parte, se puede usar para leer y escribir el contenido de un ***ArrayBuffer*** de forma no estructurada. Esto es adecuado para leer nuevos formatos de archivo, que suelen estar compuestos por combinaciones heterogéneas de datos.

El código siguiente crea un ***ArrayBuffer*** e interpreta el contenido de diversas maneras:

```
// Crear un buffer de 8 bytes
var buffer = new ArrayBuffer(8);

// convertirlo en un array de UInts8 y asignar 0x05 a
cada byte
var uint8s = new Uint8Array(buffer);
for (var i = 0; i < 8; i++) {
    uint8s[i] = 5; // rellenar cada byte con 0x05
}

// Inspeccionar el array resultante
uint8s[0] === 5; // true -cada byte tiene el valor 5
uint8s.byteLength === 8; // true -hay 8 Uint8s

// Ver el mismo buffer como un array de Uint32s
var uint32s = new Uint32Array(buffer);
```

```
// Los mismos bytes sin transformar, ahora se interpretan
de
// forma distinta

uint32s[0] === 84215045 // true - 0x05050505 == 84215045
```

Con esta técnica se pueden usar las matrices tipadas para realizar tareas diversas, como la creación de valores de coma flotante a partir de conjuntos de bytes, o la creación de estructuras de datos que requieren un diseño muy específico para la eficacia o interoperabilidad entre distintos formatos de datos.

La implementación del objeto ***DataView*** requiere bastante más espacio y remitimos al lector a los enlaces indicados anteriormente.

El nuevo modo (y la palabra reservada) *Strict*

La novedad más importante es, sin duda el nuevo modo de trabajo ***strict***, que aporta solidez, consistencia y mucha más robustez al lenguaje.

Este modo de trabajo afecta a la forma de manejar las variables desde su declaración. Ahora existen dos modos: ***default***, en el que se permite hacer referencia a variables sin declarar (al estilo utilizado hasta ahora), y que permite acciones como cambiar de tipo de dato a una variable de una instrucción a la siguiente, y otras "características" por el estilo, y ***strict*** en el que se obliga a la declaración, y en el que no están permitidas muchas de esas prácticas.

La recomendación, de cara al futuro es utilizar la segunda opción, ya que ofrece una cobertura y unos fundamentos mucho más acordes con los principios actuales de la programación. Pero debemos tener en cuenta ciertos factores.

La Sintaxis y sus contrapartidas

Sintácticamente se utiliza mediante la declaración `"use strict";`, que aplica estos principios según el ámbito de declaración: si es la primera instrucción de un fichero, o simplemente, está fuera de una función, se

asume que el fichero entero es tratado así, mientras que la declaración dentro de una función le hace local a la función.

El problema de la utilización con ámbito global, tienen que ver con lo que MVC 4 denomina "*bundling and minifying*" (empaquetado y minimización). En MVC 4 es posible hacer referencia a un directorio que contenga un bloque de ficheros, que serán empaquetados como si fueran uno solo, y de los que se eliminan los comentarios, espacios, etc., para permitir una carga más óptima por el navegador. Si utilizamos ***strict*** en la primera de sus formas, puede entenderse que todos los ficheros se ven afectados por esta característica, pudiendo producir resultados no deseados.

Futuras palabras reservadas

Un conjunto de palabras han sido declaradas como "futuras palabras reservadas" puesto que pertenecen a extensiones que ya están en estudio y no está permitido usarlas como identificadores cuando se utiliza el modo **strict**. En concreto, las siguientes:

• ***class***	• ***extends***	• ***interface***	• ***protected***
• ***const***	• ***import***	• ***let***	• ***public***
• ***enum***	• ***super***	• ***package***	• ***static***
• ***export***	• ***implements***	• ***private***	• ***yield***

Y parece que alguna va a implementarse ya en la próxima versión, como ***let***.

El uso de "***strict*"** tiene otra serie de implicaciones importantes respecto al lenguaje, que deben ser tenidas muy en cuenta, especialmente de cara a las aplicaciones web.

Otras implicaciones del modo *strict*

Entre las más importantes, podemos citar las siguientes:

- Ya no hay más variables globales implícitas dentro de una función

- La palabra reservada ***this*** ya no se vincula al objeto global cuando se llama desde una función, sino a ***undefined***.
- ***apply*** y ***call*** no se aplican al objeto global de forma predeterminada
- No hay sentencia ***with***
- No se permiten modificaciones en propiedades declaradas como no modificables o configurables (***writeable: false***, o ***configurable: false***).
- Restricciones de seguridad entorno a ***eval*** que ahora se ejecuta en su propio contexto de ejecución. Además, junto a ***arguments*** ahora son palabras reservadas.
- ***arguments*** ya no está asociada con ***parameters.*** Además, desaparecen ***arguments.caller*** y ***arguments.callee***
- Ya no existen literales de tipo octal
- Los nombres duplicados en un *Object Literal* ahora generan un error sintáctico.
- Si no se usa el prefijo ***new*** obtenemos un error, al crear objetos desde un prototipo.
- Si queremos añadir un procedimiento de evento (***eventlistener***) debemos hacerlo explícitamente sobre el objeto *window* y usaremos un prototipo como este:

```
window.addEventListener("tipo", EventListener, [Boolean
useCapture]);
```

La detección el modo *strict*

Una solución para determinar la presencia de este modo, es la sugerida por el propio Crockford basada en dos aspectos distintos de la implementación en los navegadores del modo ***strict***: cómo detectar si estamos en ese modo, y cómo detectar si el navegador soporta ese modo.

Respectivamente, podríamos escribir las siguientes funciones para

realizar esa labor:

```
function esModoEstricto() {
    return (function () {
        return !this;
    } ());
}

function modoEstrictoImplementado() {
    return (function () {
        'use strict';
        return !this;
    } ());
}
```

Novedades Sintácticas

En el apartado sintáctico hay muchas novedades que deben tenerse en cuenta al usarse el modo estricto:

- Ahora se permite el uso de comas al final de una instrucción. Por ejemplo, la longitud del *array* `[ "Coma", "final", ]` será 2 y no 3.
- No hay restricciones de palabras reservadas al asignar nombres de propiedades. Esto quiere decir que la siguiente declaración es perfectamente válida (aunque no recomendable):

```
var x = {
    return:     true,
    function:   LaFuncion,
    class:      'LaClase'
};
```

- Se permite la existencia de *getters* y *setters* al estilo C#. Por tanto podemos controlar ahora las asignaciones y los valores devueltos mediante una sintaxis familiar para los programadores de .NET, que

permite, además, muchas posibilidades. Lo veremos un poco más adelante con un ejemplo.

- ***Infitity, NaN*** y ***undefined*** ahora son constantes (no variables)
- ***parseInt*** está arreglado, de forma que ahora una llamada a ***parseInt('01')*** devuelve 1, y no 0 como hasta ahora.
- Los literales de expresiones regulares actúan ahora como funciones, produciendo un nuevo objeto ***RegExp*** cada vez que son evaluados.
- Si modificamos los objetos ***Object*** o ***Array***, no hay efectos secundarios sobre el funcionamiento de expresiones que implican llaves - **{}** - o corchetes - **[]** -.
- Si intentamos definir dos veces la misma propiedad, genera un error, y lo mismo sucede si pasamos dos veces el mismo nombre de variable.

Comprobación del modo *strict*

Una forma de saber si nuestro código cumple con este modo son las pruebas externas, como las que suministra el sitio especialmente dedicado llamado "*Try Strict*"[80] donde podemos probar nuestro código y ejecutarlo *on-line* para comprobar los cambios introducidos por esta directiva.

El sitio dispone de ejemplos prototipo ya preparados, que –al ejecutarse- generarán un mensaje de error cuando se mantenga la definición del modo estricto al comienzo del código. También podemos, copiar y pegar nuestro propio código para evaluarlo.

En general, el modo estricto es recomendable en casi todos los casos, y especialmente en las aplicaciones de cierta envergadura, donde las buenas prácticas pueden minimizar los errores sensiblemente.

Propiedades de Acceso (*getters* y *setters*)

Hasta ésta versión las propiedades eran del tipo de asignación directa,

[80] http://ie.microsoft.com/testdrive/HTML5/TryStrict/default.html

como hemos estado viendo hasta ahora. En JavaScript 5, disponemos (como apuntábamos al principio) de ***setters*** (mecanismos de asignación) y ***getters*** (mecanismos de lectura), muy al estilo de los lenguajes disponibles en .NET Framework.

Cada ***get*** o ***set*** llama a una función que devuelve lo establecido por el usuario.

Sintácticamente, el aspecto es el siguiente:

```
var Persona = {
    nombre:    'Juan',
    apellidos: 'Marcos',
    edad: 7,

    // () → String
    // Devuelve el nombre completo.
    get nombre_completo() {
        return this.nombre + ' ' + this.apellidos },

    // (nuevo_nombre:String) → undefined
    // Asigna los componentes del objeto,
    // a partir del nombre completo.
    set nombre_completo(nuevo_nombre) {
        var partesNombre;
         // Dividimos el valor fragmentos
        partesNombre = nuevo_nombre.trim().split(/\s+/)
        this.nombre = partesNombre['0'] || ''
        this.apellidos = partesNombre['1'] || ''
    }
}

alert(Persona.apellidos);  // => Marcos
alert(Persona.nombre_completo); // => Juan Marcos
Persona.nombre_completo = 'Irene García';
alert(Persona.apellidos);  // => García
```

En este código, el objeto literal *Persona* (*Object Literal*), se define con 3 propiedades tipo estándar, y una cuarta propiedad llamada ***nombre_completo*** que adopta la forma de mecanismo de acceso en

escritura (***set***) y lectura (***get***). Esta propiedad se define mediante dos funciones del mismo nombre que la propiedad, que serán llamadas al asignar desde código externo a la función la propiedad ***nombre_completo***, o al leer su valor (los comentarios finales del código indican los valores de salida en *Page Inspector*).

JavaScript 5 y *Reflection*

De nuevo nos encontramos con otra característica propia de lenguajes del estilo de .NET aquí. Las nuevas API de esta versión del lenguaje, permiten interrogar a cualquier objeto para que nos dé una descripción de cómo está definido, en forma similar a cómo las técnicas de ***Reflection*** permiten hacerlo en lenguajes como C# o VB.NET.

Por ejemplo, para saber cómo está definida una propiedad de un objeto, podemos utilizar lo siguiente, desde la solapa ***Console*** de la herramienta de depuración (F12):

```
var o = { numero: 9 }
Object.getOwnPropertyDescriptor(o, 'numero');
```

Y la consola nos mostrará esta salida:

```
Object.getOwnPropertyDescriptor(o, 'numero');
{
      value: 9,
      writable : true,
      enumerable : true,
      configurable : true
}
```

Como se ve, cada propiedad tiene atributos que definen no solo su valor (antes solo existían los "*Data Properties*"), sino también su comportamiento. En este caso, la propiedad no es de solo lectura, no se va a enumerar en un bucle, y no es configurable (o sea, no podríamos cambiar la propiedad ***writable*** a ***false***, por ejemplo).

De hecho, también podemos usar esta técnica para ver nuestro objeto

anterior, o para ver cómo está definido ***innerHTML*** dentro de un objeto **HTMLElement**:

```
Object.getOwnPropertyDescriptor(HTMLElement.prototype,
'innerHTML');
```

Y veremos cómo en la salida se nos muestra como una propiedad ***get/set***:

```
 Object.getOwnPropertyDescriptor(HTMLElement.prototype,
'innerHTML');
{
      get :  function innerHTML() { [native code] } ,
      set :  function innerHTML() { [native code] } ,
      enumerable : true,
      configurable : true
}
```

Y, como es lógico, el atributo ***writable*** ha sido sustituido por el par de funciones ***get/set*** que permiten dinámicamente la modificación del elemento. (Nótese que, en este caso, estamos accediendo a una definición del DOM y no del lenguaje).

Dado que tenemos también una API para modificar el descriptor de una propiedad cualquiera -***defineProperty()***-, esto plantea una situación interesante, en la que podemos intervenir dinámicamente sobre definiciones del DOM y modificar su comportamiento, en lo que podría asemejarse a un proceso de "sub-clasificación".

Novedades en IE11

Internet Explorer 11 incorpora ya algunas de las novedades que se apuntan para la versión ECMAScript 6 (que, según algunos de sus promotores, estará disponible a finales de 2013).

De esta forma, el equipo de desarrollo se alinea con el trabajo del grupo TC39 de ECMA (el encargado de esta especificación), y –tal y como anunciaron en su momento- van a ir incorporando en las actualizaciones

del navegador las novedades propuestas tan pronto como estén disponibles.

Las que introduce IE11, pueden dividirse en 4 categorías: Declaraciones de ámbito de variables, Objetos contenedores, API de internacionalización y la propiedad ***_proto_***. Vamos a verlas someramente, ya que su implementación no está todavía completa, si bien debemos de tener en cuenta que estas novedades funcionan, tanto en el explorador como en Aplicaciones Windows 8 hechas con HTML5 y JavaScript.

Declaraciones de ámbito en variables

Ahora podemos utilizar las palabras clave ***let*** y ***const*** para declarar variables en las que el alcance se limita a la sentencia en la que están declaradas.

Mediante ***let***, podemos declarar variables con la garantía de que su ámbito de declaración definirá su visibilidad. El ejemplo oficial, (adaptado) que aparece en la página MSDN de Microsoft lo plantea de la siguiente forma:

```
var  l = 10;
{
    let l = 2;
   // En este punto, l = 2.
}
// Y aquí, l = 10.

// Otras formas de declarar una variable mediante let.
let index;
let name = "Thomas Jefferson";
let answer = 42, counter, numpages = 10; // counter =
undefined
let myarray = new Array();
```

Exactamente igual, y con una sintaxis prácticamente idéntica, podemos declarar constantes mediante la palabra reservada ***const***.

El ejemplo equivalente al anterior resulta bastante explicativo por sí mismo:

```
var  c = 10;
{
    const c = 2;
   // En este punto, c = 2.
}
// Y aquí, c = 10.

// Otras formas de declarar una variable mediante let.
const name = "Thomas Jefferson";
const answer = 42, numpages = 10;
const myarray = new Array();
```

Con esto evitamos tener que recurrir a algunas de las técnicas de "*closing*" que hemos indicado más arriba para preservar el ámbito de variables y constantes.

Objetos contenedores

También hacen su aparición 3 nuevos objetos que nos permiten crear colecciones de elementos únicos (esto es, que no pueden repetirse dentro de la colección). Y esto es válido tanto para objetos de cualquier clase, como para colecciones de tipo clave/valor en dos variantes distintas.

Para definirlos, por un lado tenemos la sentencia objeto ***Set*** y para las variantes de parejas, los objetos ***Map*** y ***WeakMap***.

El objeto *Set*

Mediante el primero, podemos crear una especie de Array de tipo múltiple, del cual poder extraer posteriormente su información mediante referencias directas o mediante un bucle, como muestra el ejemplo oficial de MSDN, que utilizamos aquí después de una adaptación:

```
var s = new Set();
s.add("Thomas Jefferson");
```

```
s.add(1776);
s.add("founding father");

s.forEach(function (item) {
    document.write(item.toString() + ", ");
});

// Salida:
// Thomas Jefferson, 1776, founding father,
```

Además de su constructor y su prototipo, este objeto posee una propiedad ***size*** que nos indica el total de objetos que alberga en un instante dado.

Y, como se ve en el código fuente adjunto, dispone de métodos de inserción de un objeto (***add***) y de recorrido (***forEach***), además de los métodos ***clear*** (para borrar todo el contenido), ***delete*** (para borrar un elemento) y ***has*** (comprobar la presencia de un elemento).

Los objetos *Map* y *WeakMap*

Por su parte, los objetos ***Map*** y ***WeakMap*** poseen exactamente las mismas propiedades y métodos que el anterior, cambiando únicamente los valores que almacenan y la forma en que se accede a ellos.

El código siguiente crea un nuevo objeto, lo rellena con 4 pares distintos y lo recorre mostrando la salida.

```
var m = new Map();
m.set(1, "black");
m.set(2, "red");
m.set("colors", 2);
m.set({x:1}, 3);

m.forEach(function (item, key, mapObj) {
    document.write(item.toString() + "<br />");
});

document.write("<br />");
document.write(m.get(2));
```

```
// Salida:
// black
// red
// 2
// 3
//
// red
```

Como puede verse en la última instrucción, la forma de acceso externa al bucle es mediante el método ***Get(índice)***, mientras que la forma de asignación es mediante ***Set(clave, valor).***

En estos objetos, cuando añadimos un elemento a la colección y su clave ya existe, el recién insertado sustituye al existente.

La diferencia más notable con los objetos ***WeakMap*** es que éstos no son enumerables. Y es que, en un objeto ***WeakMap***, las referencias a los objetos principales se declaran como 'débiles'.

Esto significa que ***WeakMap*** no evitará una recolección de basura de los objetos clave. Téngase en cuenta que cuando no existen referencias (aparte de las indicadas por un ***WeakMap***) a los objetos clave, el recolector de basura puede destruir esos objetos clave.

El código oficial que nos muestra este comportamiento es el siguiente:

```
var dog = {
    breed: "yorkie"
}

var cat = {
    breed: "burmese"
}

var wm = new WeakMap();
wm.set(dog, "fido");
wm.set(cat, "pepper");

document.write(wm.get(dog) + ": ");
```

```
document.write(dog.breed);
document.write("<br />");
document.write(wm.get(cat) + ": ");
document.write(cat.breed);

// Salida:
// fido: yorkie
// pepper: burmese
```

Obsérvese que la asignación sigue realizándose mediante ***set(clave, valor)*** y la recuperación de los datos, mediante ***get(clave)***.

La API de internacionalización

Suministra al desarrollador herramientas que permiten formatear fechas, valores numéricos, y comparaciones, tal y como se especifica en la **ECMAScript Internationalization API Specification**.

Está basada en 3 nuevos objetos: ***Intl.DateTimeFormat*** **Object**, ***Intl.NumberFormat Object*** e ***Intl.Collator Object***.

En todos los casos, los métodos encargados de devolver el valor adaptado a la nacionalidad correspondiente, reciben como argumento un "*locale*"[81] indicando el idioma y la región correspondientes, como ya hemos comentado en el capítulo dedicado a HTML.

Para más datos sobre su programación, recomendamos una visita al sitio Web oficial que incluimos en el apartado de Referencias. De momento, podemos darnos una idea de su funcionamiento con el primer ejemplo oficial adaptado:

```
a
var date = new Date(Date.UTC(2013, 1, 1, 14, 0, 0));
var options = { weekday: "long", year: "numeric", month:
"short",
    day: "numeric" };
```

[81] Se utiliza los identificadores de lenguaje definidos por el BCP47, disponibles en el documento http://tools.ietf.org/html/rfc5646

```
if (console && console.log) {
    console.log(new Intl.DateTimeFormat("en-
US").format(date));
    // Devuelve 2/1/2013
    console.log(new Intl.DateTimeFormat("ja-
JP").format(date));
    // Devuelve 2013年2月1日
    console.log(new Intl.DateTimeFormat("ar-SA",
options).format(date));
    // Devuelve الجمعة, ٢٠ ربيع الأول, ١٤٣٤
    console.log(new Intl.DateTimeFormat("hi-IN",
options).format(date));
    // Devuelve शुक्रवार, 01 फरवरी 2013
}
```

Para el caso de los números el objeto ***NumberFormat*** recibe igualmente un *locale*, pero, además dispone de la posibilidad de que se le indiquen opciones extra (por ejemplo para porcentajes, símbolos de moneda, etc.).

Podemos verlo claramente en los dos ejemplos que nos ofrece el sitio oficial del MSDN:

```
var nf = new Intl.NumberFormat(["en-US"], {
    style: "currency",
    currency: "CNY",
    currencyDisplay: "symbol",
    maximumFractionDigit: 1
});

if (console && console.log) {
    console.log(nf.format(100)); // Returns ¥100.00
}
```

Aquí, vemos que el segundo argumento de ***NumberFormat*** admite un objeto donde podemos especificar opciones como la moneda, tipo de símbolo a devolver, el estilo y el número de decimales a usar.

En el siguiente ejemplo, se utiliza una mezcla de ambos para devolver datos de distintos *locale*, usando las mismas opciones predefinidas:

```
var number = 123456789;
var options1 = { style: "percent" };
var options2 = { style: "currency", currency: "INR" };

if (console && console.log) {
    console.log(new Intl.NumberFormat("en-
US").format(number));
    // Returns 123,456,789
    console.log(new Intl.NumberFormat("ja-
JP").format(number));
    // Returns 123,456,789
    console.log(new Intl.NumberFormat("ar-SA",
options1).format(number));
    // Returns ١٢,٣٤٥,٦٧٨,٩٠٠ %
    console.log(new Intl.NumberFormat("hi-IN",
options2).format(number));
    // Returns ₹ 12,34,56,789.00
}
```

El tercer objeto de la colección ***Intl*** es ***Intl.Collator*** y sirve para realizar comparaciones de cadenas (especialmente para establecer criterios de ordenación en salidas), teniendo en cuenta la sintaxis especial del *locale* que se indique en la creación del objeto ***Collator.***

Al igual que *NumberFormat,* ***Collator*** admite dos parámetros opcionales: un *locale* y otro donde se puede indicar opciones de comparación. El ejemplo oficial nos da una idea de su funcionamiento:

```
var co1 = new Intl.Collator(["de-DE-u-co-phonebk"]);
var co2 = new Intl.Collator(["de-DE"]);
var co3 = new Intl.Collator(["en-US"]);

var arr = ["ä", "ad", "af", "a"];

if (console && console.log) {
```

```
    console.log(arr.sort(co1.compare));  // Devuelve
a,ad,ä,af
    console.log(arr.sort(co2.compare));  // Devuelve
a,ä,ad,af
    console.log(arr.sort(co3.compare));  // Devuelve
a,ä,ad,af
}
```

También se puede usar para buscar cadenas donde existen caracteres especiales, como vemos en el otro ejemplo oficial:

```
// String to search
var arr = ["ä", "ad", "af", "a"];
// String searched for
var s = "af";

var co = new Intl.Collator("de-DE", { usage: "search" });
var matches = arr.filter(function (i) {
    return co.compare(i, s) === 0;
});

if (console && console.log) {
    console.log(matches);  // Returns af
}
```

La propiedad *_proto_*

Como hemos visto en este mismo capítulo, en JavaScript, el concepto de prototipo es importante (y necesario de conocer) si queremos realizar ciertas operaciones con un objeto.

Con la idea de facilitar este tipo de operaciones, IE11 (y el estándar ECMAScript 6) pone a disposición esta propiedad, que siempre almacena una referencia al prototipo interno del objeto sobre el que se aplique.

Mediante esta propiedad podremos establecer un nuevo prototipo para un objeto dado de forma sencilla. El objeto o la función a la que se aplique, heredan todos los métodos y propiedades del nuevo prototipo, junto con todos los métodos y propiedades de la cadena de prototipo

del nuevo prototipo.

Un objeto puede tener un solo prototipo (sin incluir prototipos heredados en la cadena de prototipo), por lo que cuando se llama a la propiedad **__proto__**, se reemplaza el prototipo anterior.

En el siguiente ejemplo oficial vemos como cambiar el prototipo de un objeto:

```
function Rectangle() {
}

var rec = new Rectangle();

if (console && console.log) {
    console.log(rec.__proto__ === Rectangle.prototype);
// Devuelve  true
    rec.__proto__ = Object.prototype;
    console.log(rec.__proto__ === Rectangle.prototype);
// Devuelve false
}
```

También podemos añadir propiedades a un objeto, si se las añadimos a su prototipo:

```
var proto = { y: 2 };

var obj = { x: 10 };
obj.__proto__ = proto;

proto.y = 20;
proto.z = 40;

if (console && console.log) {
    console.log(obj.x === 10);  // Devuelve true
    console.log(obj.y === 20);  // Devuelve true
    console.log(obj.z === 40);  // Devuelve true
}
```

Como esta característica está presente para cualquier objeto, también

podemos hacer lo mismo con los objetos predefinidos por JavaScript, como ***String***:

```
var stringProp = { desc: "description" };

String.__proto__ = stringProp;
var s1 = "333";
var s2 = new String("333");

if (console && console.log) {

    console.log(String.desc === "description");
// Devuelve true
    console.log(s1.desc === "description");
// Devuelve false
    console.log(s2.desc === "description");
// Devuelve false

    s1.__proto__ = String;  // Can't be set.
    s2.__proto__ = String;

    console.log(s1.desc === "description");
// Devuelve false
    console.log(s2.desc === "description");
// Devuelve true
}
```

Como hemos visto en este repaso por el lenguaje JavaScript, se trata de un lenguaje con características dinámicas muy interesantes, y que tiene una presencia como muy pocos a nivel mundial, por lo que todas las novedades que presente son muy a tener en cuenta por cualquier desarrollador.

En el capítulo siguiente, veremos las API que define el estándar y que suponen la programación de opciones añadidas a las páginas y/o sitios Web programadas íntegramente con JavaScript.

TypeScript

Pero, como anticipábamos más arriba, ha hecho aparición un nuevo protagonista que pretende facilitar la labor de los desarrolladores, y, en especial, la de aquellos que tienen que realizar aplicaciones de cierto calado usando este lenguaje.

Me refiero, naturalmente, a ***TypeScript***, la última creación del autor de C# y arquitecto principal de .NET, **Anders Hejlsberg**, quien, como siempre, consciente de los retos a los que se enfrentaban estos perfiles de desarrollo ha creado una sintaxis basada en JavaScript, que "compila" a código JavaScript, pero que dispone de muchos de los recursos a los que los desarrolladores de lenguajes orientados a objetos estamos acostumbrados: clases, interfaces, módulos, ámbito de variables, y un interesante etcétera, que permite que el abordaje de una gran aplicación no sea tan duro de realizar (No olvidemos que JavaScript jamás se diseñó con el objetivo para el que está siendo utilizado hoy en día).

Durante el evento BUILD/2013 de San Francisco, Microsoft presentó la última versión de este lenguaje (la 0.9) mucho más madura y utilizable que las anteriores y perfectamente integrada dentro del entorno de Visual Studio (2012/2013).

TypeScript se presenta como una sintaxis editable desde cualquier editor de texto, si bien bajo Visual Studio se nos presenta como un complemento instalable y descargable desde la página dedicada "oficial"82, y también como parte de la instalación de la librería Node.js.

Igualmente, existe una página perteneciente al sitio oficial (la página Playground[83]), que permite probar "on-line" diversos ejemplos predefinidos, así como cualquier código que el usuario quiera testar.

---

[82] http://www.typescriptlang.org/#Download

[83] http://www.typescriptlang.org/playground/

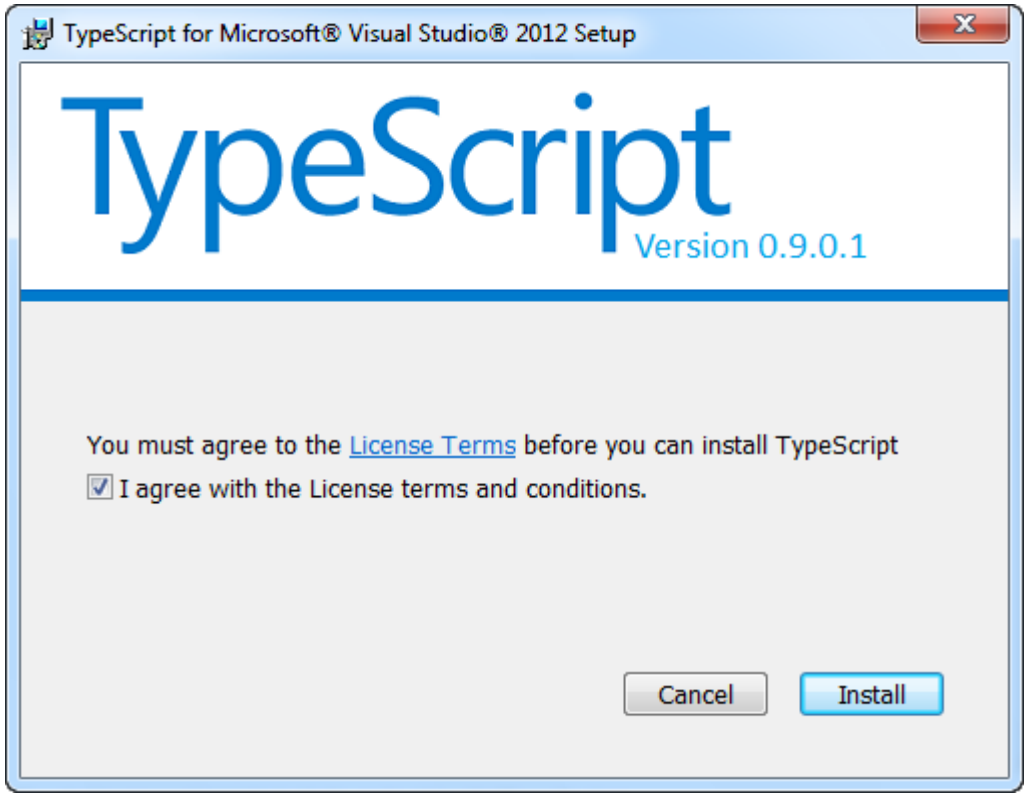


Fig. 5: Ventana inicial de la Instalación de TypeScript

Una vez instalado el complemento, disponemos de un nuevo tipo de fichero editable en el que los conceptos definidos por esta sintaxis JavaScript son perfectamente válidos y reconocibles por el editor: conceptos como clase, interfaz, módulo, ámbito de variables y unos cuantos más.

Desde Visual Studio 2012, observamos el resultado de la instalación del como una entrada más dentro del apartado "Extensiones y Actualizaciones", como podemos ver en la Figura 6.

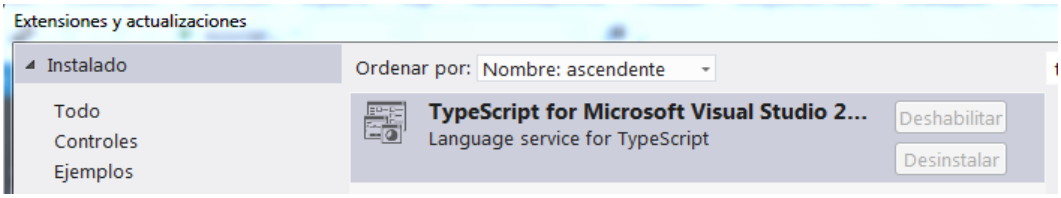


Fig. 6: El complemento TypeScript desde Visual Studio 2012.

TypeScript se anticipa así a las ofertas que platea presentar la próxima versión del estándar JavaScript (ECMAScript 6.0), que se encuentra en estos momentos en fase de desarrollo, y con el que están colaborando muchos de los principales protagonistas de la industria.

Utilizando esta sintaxis, es posible definir código como el que aparece en la siguiente captura de pantalla, en el que definimos una clase Persona, que contiene un método NombrePersona (de tipo string), que devuelve una cadena compuesta de los campos Nombre y Apellido formateados:

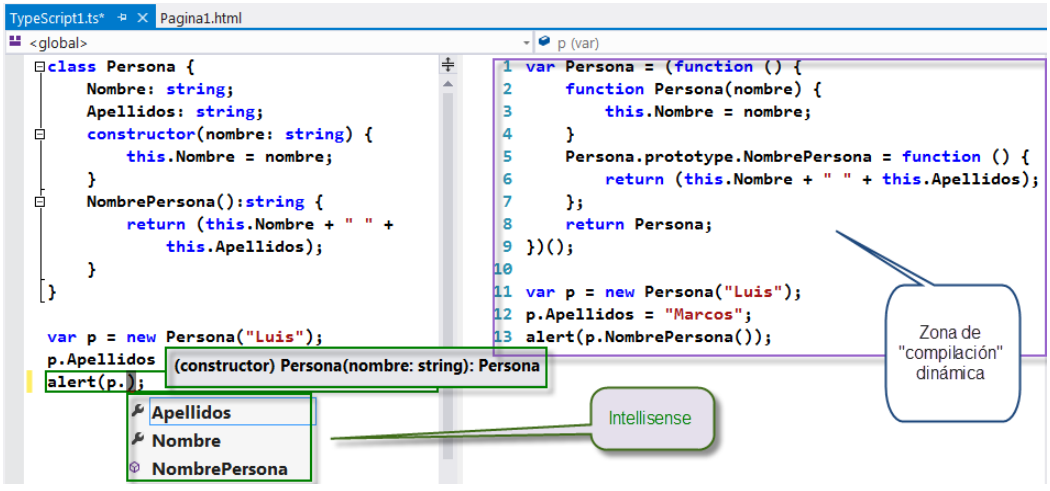


Fig. 7: Editor de Visual Studio mostrando un código TypeScript, la ventana de "compilación" y el soporte de "Intellisense"

Como vemos en la figura, según estamos editando el código TypeScript (ventana izquierda), Visual Studio nos ofrece (cada vez que guardamos el código fuente) la versión del mismo código en JavaScript en la ventana

de "compilación dinámica", al tiempo que soporta las habituales opciones de "Intellisense" normales en cualquier otro lenguaje .NET. Como resultado del proceso de compilación, Visual Studio genera 3 archivos: el original de extensión .ts, y dos JavaScript: la versión original formateada, .js y una versión minimizada para ser usada en producción por razones de rendimiento, de extensión .min.js.

Para probar el código, dentro del fichero HTML, podemos hacer una referencia al propio fichero original (.ts), a su "compilado" (.js), e incluso o a su versión minimizada (.min.js). Aunque, obviamente, en la versión de producción utilizaremos la versión minimizada por razones de rendimiento en los tiempos de carga.

En el apartado de la edición y distribución de código CSS y HTML, también encontramos novedades interesantes en esta versión, así como desde el punto de vista de la depuración, con el que ya se dio un paso importante en la versión anterior al presentar la herramienta Page Inspector.

Referencias

- "*JavaScript: The definitive Guide*". 6ª Edición. **David Flanagan**. O'Reilly Media, 2011.
- "*JavaScript: The Good Parts*". **Douglas Crockford**. O'Reilly Media, 2008.
- Guía de Internet Explorer 10 para desarrolladores: http://msdn.microsoft.com/es-es/library/ie/hh673549(v=vs.85)
- *Internet Explorer 11 Preview guide for developers* (JavaScript): http://msdn.microsoft.com/en-us/library/ie/dn342892(v=vs.85).aspx
- *ECMAScript Internationalization API Specification*: http://www.ecma-international.org/ecma-402/1.0/
- *ECMAScript 6 modules: the future is now:* http://www.2ality.com/2013/07/es6-modules.html

- *Toward Modern Web Apps with ECMAScript 6*: http://www.sencha.com/blog/toward-modern-web-apps-with-ecmascript-6/

Capítulo 06 | Las API de JavaScript

Antes de nada, debemos aclarar qué entendemos –o mejor, qué se entiende habitualmente- por una API de JavaScript. Hemos visto que en la parte HTML existen elementos que son programables. O sea, que "exponen" una API programable para que el desarrollador pueda complementar su funcionalidad de acuerdo a una necesidad.

Ese era el caso del elemento **<*input*>** respecto a las validaciones, o –en el apartado de los atributos- de la programación de expresiones regulares, o los menús contextuales, por ejemplo. Recordemos que esto es posible debido a la existencia de una interfaz denominada

Document Object Model (DOM) Level 2 HTML Specification[84], que provee un conjunto de propiedades y métodos que permiten crear y/o manipular la estructura de un documento HTML. Aunque, en principio esta especificación es independiente del lenguaje, todos los navegadores utilizan ECMAScript (JavaScript), para permitir el acceso mediante programación a estos elementos.

El conjunto completo de estas API disponibles para la versión HTML 5 y actualizados con fecha 6/Agosto/2013 se encuentra disponible en un formato más legible para los programadores en el documento oficial "***HTML5. A vocabulary and associated APIs for HTML and XHTML***"[85].

Por otra parte, existe un amplio conjunto funcional que no es posible asociar –en principio- a ningún elemento del DOM y que debe ser programado de forma separada, mediante JavaScript. A este grupo pertenecen algunas de las API ya existentes y/o renovadas, como AJAX (*XHR*, para ser exactos), *Navigation Timing* u *Offline Applications,* y por otro lado, todo el bloque de nuevas API que pertenecen a esta versión HTML5, y que están en distintas niveles de terminación en estos momentos.

Naturalmente, el análisis de cada una de las API que están en progreso iría mucho más allá del ámbito de este libro, por lo que hemos optado por analizar aquellas que por su adopción e implementación se destacan como las más importantes en los nuevos sitios y aplicaciones Web que se basan en HTML5.

Téngase en cuenta que la clasificación oficial por categorías incluye los apartados: ***Standards*** (totalmente concluidos), ***Group Notes*** (concluidos, pero pendientes de la aprobación final de W3C), ***Candidate Recommendations*** (en fase de aprobación final), ***Last Call*** (última llamada a la comunidad para hacer cambios) y ***Working***

[84] http://www.w3.org/TR/2003/REC-DOM-Level-2-HTML-20030109/

[85] http://www.w3.org/TR/2013/CR-html5-20130806/

Drafts (borradores de trabajo en fase de desarrollo).

No obstante, para que el lector pueda darse una idea del estado de la situación, listamos a continuación todas las API de JavaScript que están siendo desarrolladas por la W3C, con una breve descripción de su propósito (mantenemos el nombre del API en su original en inglés).

Calificados como *Standards*

Son las especificaciones ya concluidas y, en su inmensa mayoría de aspectos, totalmente implementadas en los navegadores modernos. Hemos visto el uso de Selectores en el capítulo de CSS3, y veremos el API Web Storage. Implícitamente, hemos estado usando casi todas ellas a lo largo de este libro.

Piense el lector que, algunas de estas API son el fundamento de características que hemos visto en las herramientas de depuración de los navegadores (y en ocasiones del propio V. Studio).

Web Storage
Esta especificación define una API para el almacenamiento persistente de datos en pares clave-valor en los clientes Web.

Page Visibility
Esta especificación define un medio para que los desarrolladores de sitios puedan determinar mediante programación el estado de visibilidad actual de la página con el fin de desarrollar aplicaciones web eficaces respecto a las fuentes de consumo y gestión de la potencia de CPU.

Selectors API Level 1
Los selectores, que son ampliamente utilizados en CSS, son patrones que coinciden con los elementos en una estructura de árbol. La especificación de los selectores API define métodos para recuperar nodos Element del DOM, haciéndolo coincidir con un grupo de selectores. A menudo es deseable llevar a cabo las operaciones de DOM en un conjunto específico de elementos en un documento.

Estos métodos simplifican el proceso de adquisición de elementos específicos, sobre todo en comparación con las técnicas más detalladas definidas y utilizadas en el pasado.

Navigation Timing
Este documento proporciona una interfaz para aplicaciones web para acceder a la información temporal relacionada con la navegación y los elementos del DOM.

High Resolution Time
Esta especificación define una interfaz que proporciona la hora actual en la resolución sub-milisegundos de tal manera que no está sujeto a imprecisiones del reloj del sistema u otros ajustes.

Element Traversal Specification
Esta especificación define la interfaz ElementTraversal, que permite a los scripts navegar por los elementos de un árbol DOM, con exclusión de todos los demás nodos del DOM, como nodos de texto. También proporciona un atributo para exponer el número de elementos secundarios de un elemento. Se pretende proporcionar una alternativa más conveniente a las interfaces de navegación DOM existentes.

Calificados como *Group Notes*

En este bloque, se incluyen algunas API ya existentes que se han revisado y actualizado (como todo lo relativo a RDF), y también otras que tienen ya mucho tiempo de existencia, pero que no han sido ampliamente aceptadas e implementadas en los navegadores hasta ahora como las aplicaciones "*off-line*", que veremos en este capítulo.

Web Intents
Esta especificación define un mecanismo para el descubrimiento de servicios y llamadas RPC ligeras a aplicaciones web llamado **Web Intents.**

Este documento define las interfaces DOM y etiquetas utilizadas por las páginas de servidor y de cliente para crear, recibir y responder a mensajes Web Intents, así como los procedimientos que el agente de usuario lleva a cabo para facilitar ese proceso.

Web Audio Processing: Use Cases and Requirements

Este documento presenta una serie de escenarios y una lista de requisitos que rigen la labor del Grupo de Trabajo de Audio W3C en su desarrollo de una API web para el procesamiento y síntesis de audio en la web.

Web API Design Cookbook

Este documento recoge las prácticas más comunes en el diseño de API que encajan bien en la plataforma Web en su conjunto, usando *WebIDL*.

RDFa API

RDFa permite a los autores publicar información estructurada que es a la vez humana y legible. Conceptos que tradicionalmente han sido difíciles para las máquinas para detectar, como personas, lugares, eventos, música, películas y las recetas, son ahora fácilmente marcadas en los documentos Web. Mientras que la publicación de estos datos es vital para el crecimiento de Linked Data, el uso de la información para mejorar la utilidad colectiva de la Web para la humanidad es el verdadero objetivo.

Para lograr este objetivo, debe resultar sencillo para los desarrolladores web extraer y utilizar información estructurada de un documento Web. El presente documento detalla un mecanismo de este tipo, una interfaz de programación de aplicaciones RDFa (API RDFa) que permite la simple extracción y el uso de información estructurada de un documento Web.

Web Application Privacy Best Practices
Este documento describe las mejores prácticas de privacidad para aplicaciones web, incluyendo aquellas que podrían utilizar las API de dispositivos.

Device APIs Requirements
Estos son los requisitos que se debe satisfacer en el desarrollo de las API de cliente que permiten la creación de aplicaciones Web y *widgets* Web que interactúan con los servicios de dispositivos tales como calendario, contactos, cámara, etc.

Offline Web Applications
Aplicaciones Web Offline destaca las características de HTML 5 que abordan el reto de construir aplicaciones web que funcionan sin conexión.

Borradores (Working Drafts)

Los siguientes corresponden a proyectos de documentos ***Candidate Recommendations***, ***Last Call Drafts*** y ***Working Drafts***.

Candidate Recommendations

Si bien no llegan al estado final, es muy improbable que la especificación sufra algo más que un cambio menor desde su situación actual. Hay varias que ya están implementadas ampliamente y –en este capítulo- veremos algunas de ellas por su importancia en el desarrollo, como IndexedDB, WebSockets y Web Workers; sin duda 3 de las más importantes.

Vibration API
Una API para controlar el vibrador del dispositivo.

Indexed Database API
Este documento define las API para una base de datos que contiene valores simples y objetos jerárquicos, con posibilidad de indexación,

filtros y capacidad transaccional.
HTML Media Capture Esta especificación define mejoras de los formularios HTML que proporcionan acceso a las capacidades de audio, imagen y captura de video de los dispositivos.
Pointer Events Define eventos e interfaces para el manejo de entrada de un mecanismo apuntador agnóstico al hardware desde dispositivos como el ratón, la pluma o la pantalla táctil.
Server-Sent Events Esta especificación define una API para abrir una conexión HTTP para recibir notificaciones push desde un servidor en la forma de eventos DOM.
The WebSocket API Esta especificación define una API que permite que las páginas web utilicen el protocolo de Web sockets para la comunicación dúplex con un host remoto.
Performance Timeline Esta especificación define una interfaz para aplicaciones web para acceder a la información de tiempo relacionados con la navegación y los elementos de marcado. Es utilizada por otras especificaciones, como **User Timing**.
User Timing Esta especificación define una interfaz para ayudar a los desarrolladores web a medir el rendimiento de sus aplicaciones, dándoles acceso a marcas de tiempo de alta precisión.
Resource Timing Esta especificación define una interfaz para aplicaciones web para acceder a la información de tiempo en relación con los elementos HTML.

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

Web Workers
Esta especificación define una API que permite a los autores de aplicaciones web lanzar una tarea en segundo plano, ejecutando scripts en paralelo a su página principal. Esto permite las operaciones en modo de hilo de ejecución, con traspaso de mensajes como mecanismo de coordinación.

HTML5 Web Messaging
Esta especificación define dos mecanismos para la comunicación entre contextos de navegación en el documento HTML.

Web IDL
Este documento define un lenguaje de definición de interfaz, IDL de Web, que puede ser utilizada para describir las interfaces que están destinados a ser implementadas en los navegadores web.

Progress Events
Este documento describe los tipos de eventos que se pueden utilizar para monitorear el progreso de una operación. Está destinado principalmente a contextos tales como las operaciones de transferencia de datos especificados por ***XMLHTTPRequest*** [XHR], o ***Media Access Events*** [MAE].

Borradores en estado *Last Call*

Finalmente, aquí figuran aquellas que todavía podrían recibir cambios o anulaciones importantes en su contenido.

JSON-LD 1.0 Processing Algorithms and API
Una interfaz de programación de aplicaciones y un conjunto de algoritmos para la programación de documentos JSON-LD (*JSON-based Serialization for Linked Data*) con el fin de que sean más fáciles

de usar en entornos de programación como JavaScript, Python y Ruby.

API for Media Resources 1.0
Esta especificación define una API de cliente para acceder a la información de los metadatos relacionados con los recursos multimedia en la Web.

Ambient Light Events
Esta especificación define un medio para recibir los eventos que corresponden a un sensor de lumínico que detecte de la presencia de una luz.

Proximity Events
Esta especificación define un medio para recibir los eventos que corresponden a un sensor de proximidad que detecte la presencia de un objeto físico.

Timing control for script-based animations
Este documento define un sitio web del CDE autores pueden utilizar para escribir secuencias de comandos basados en animaciones donde el agente de usuario tiene el control de limitación de la velocidad de actualización de la animación (como ***requestAnimationFrame***).

Geolocation API Specification Level 2
Esta especificación añade la posibilidad de recuperar una dirección urbana, en lugar de sus coordenadas como sucede con la API de geo-localización.

DeviceOrientation Event Specification
Esta especificación define nuevos tipos de eventos DOM que proporcionan información sobre la orientación física y el movimiento de un dispositivo que la aloja.

Otros Borradores de Trabajo

En este apartado figuran algunas de las API que comentamos en este capítulo, pero que –a pesar de no estar en un grado más avanzado de

la especificación- ya han sido implementadas, en parte o totalmente en varios navegadores. Entre ellas, veremos ***XmlHttpRequest*** y **File API**.

Input Method Editor API Define una API que proporciona aplicaciones web con acceso mediante scripting a un Editor de métodos de entrada
Push API Una API que proporciona a las aplicaciones web acceso mediante scripting a los datos enviado por un servidor de aplicaciones.
Mediastream Image Capture Esta especificación define una API para tomar imágenes fijas desde un dispositivo de captura para la generación de un flujo de vídeo.
Web Cryptography API Esta especificación describe un API de JavaScript para la realización de operaciones básicas de cifrado en las aplicaciones web, como hash, generación y verificación de firmas y el cifrado y el descifrado. Además, se describe una API para aplicaciones para generar y/o gestionar el material clave necesario para llevar a cabo estas operaciones. Se proporciona almacenamiento de claves tanto para claves temporales como permanentes. El acceso a las claves depende de la política cross-origin establecida. Algunos usos de este API incluyen servicios de autenticación, firma digital de documentos o código, así como la confidencialidad y la integridad de las comunicaciones.
Web Telephony API Esta especificación define una API para gestionar las llamadas telefónicas. Un caso típico de uso de la API de Telefonía Web es la implementación de una aplicación de 'Marcador' con soporte de varias llamadas y múltiples servicios de telefonía. También se define una estructura mínima de los elementos del historial de llamadas.

Media Capture and Streams
Este documento define un conjunto de API que permiten que los elementos multimedia locales, incluyendo audio y vídeo, reciban solicitudes de una plataforma, y puedan ser enviados a través de la red a otro navegador o dispositivo que implemente el conjunto apropiado de protocolos en tiempo real, para ser procesados y visualizados localmente.

The app: URI scheme
Esta especificación define una **App**: el esquema y las reglas para la resolución de referencias a una aplicación; La URI, se refiere a cómo hacer para referenciar los recursos dentro de un paquete (en una aplicación empaquetada). El modelo se basa en la eliminación de referencias semánticas HTTP para devolver los recursos de una manera similar a una solicitud HTTP GET. De esta forma, permite que este esquema URI pueda ser utilizado con otras tecnologías que se basan en respuestas HTTP para funcionar como se pretende, tales como [XHR].

Raw Socket API
Esta API proporciona interfaces a sockets UDP, sockets cliente TCP y sockets de servidor TCP.

Encrypted Media Extensions
Esta especificación extiende la interfaz *HTMLMediaElement* para proporcionar una API que controle la reproducción de contenido protegido.

Media Source Extensions
Esta especificación extiende la interfaz HTMLMediaElement para autorizar a código JavaScript a generar flujos multimedia para su reproducción.

Clipboard API and Events
Este documento describe las API para las operaciones de portapapeles, como copiar/cortar y pegar, o arrastrar y soltar en las aplicaciones web.

Network Service Discovery
Esta especificación define un mecanismo para un documento HTML para descubrir y posteriormente comunicarse con los servicios basados en HTTP anunciados a través de protocolos de detección comunes dentro de la red de un usuario.

Runtime and Security Model for Web Applications
Este documento define un "*Runtime*" y el modelo de seguridad para aplicaciones Web. Se describe cómo se define una aplicación a través de un manifiesto de aplicación, y la forma en que se puede instalar, actualiza y se empaquetado. También especifica cómo una aplicación se puede poner en segundo plano, traerse de nuevo a primer plano o ser "despertada" de un estado de hibernación.

Por último, el documento describe el modelo de seguridad para este tipo de aplicaciones. Esto incluye el modelo de permisos y las diferentes normas de seguridad que se aplicarían.

Messaging API
Esta especificación define una API a nivel de sistema que ofrece una interfaz sencilla para obtener el acceso a los servicios de mensajería móvil. Un caso típico de uso de la API de mensajería es la implementación de una aplicación de cliente de mensajería que permite al usuario enviar mensajes SMS y MMS, así como para acceder y administrar los mensajes SMS y MMS recibido.

WebDriver
Esta especificación define la API WebDriver, una interfaz neutro a la plataforma y al lenguaje que permite que los programas o scripts analizar ciertos aspectos y controlar el comportamiento de un navegador web.

Contacts Manager API
Un API de nivel de sistema para la gestión de los contactos del usuario que están almacenados en la libreta de direcciones del sistema.

Streams API
Define una API para representar datos binarios en aplicaciones web como un objeto *Stream*.

MediaStream Recording
Este documento define una API para la grabación de audio y secuencias de vídeo.

Web Alarms API Specification
Esta especificación define una API de nivel de sistema para proporcionar acceso a los ajustes de la alarma del dispositivo, que puede programar una notificación o para que una aplicación se inicie a una hora específica.

Por ejemplo, algunas aplicaciones como el reloj despertador, el calendario o la actualización automática, tienen que utilizar la API de alarma para activar determinados comportamientos del dispositivo en los puntos de tiempo especificados.

Navigation Timing 2
Este documento proporciona una interfaz para aplicaciones web para acceder a la información de tiempo relacionada con la navegación y los elementos de marcado.

Web Audio API
Esta especificación describe una API JavaScript de alto nivel para el procesamiento y síntesis de audio en aplicaciones web. El paradigma primario es el de un grafo de enrutamiento de audio, donde un número de objetos *AudioNode* están conectados entre sí para definir la representación general de audio. El procesamiento real se llevará a cabo principalmente en la implementación subyacente (por lo general optimizado en Ensamblador/C/C ++), pero también soporta el procesamiento y síntesis con JavaScript.

XMLHttpRequest
La especificación ***XMLHttpRequest*** define una API que proporciona la funcionalidad de scripting de cliente para la transferencia de datos entre un cliente y un servidor, uno de los

componentes básicos de "AJAX".

The Screen Orientation API
Define las API para leer el estado de orientación de la pantalla y para bloquear la orientación de la pantalla a un estado específico.

The Network Information API
El API ***Network Information*** proporciona una interfaz para aplicaciones web para acceder a la información de la red subyacente (Información de conexión) en el dispositivo.

File API
Esta especificación proporciona una API para la representación de objetos ***File*** (archivo) en las aplicaciones web, así como para su programación, seleccionándolos y accediendo a sus datos.

Web Intents Addendum - Local Services
Esta especificación extiende el servicio de descubrimiento de ***Web Intents*** mediante la definición de extensiones opcionales que habilitan a los agentes de usuario a descubrir y registrar dinámicamente otros servicios *Web Intents* locales.

WebRTC 1.0: Real-time Communication Between Browsers
Este documento define un conjunto de API que permiten que los elementos multimedia locales, incluyendo audio y vídeo, puedan recibir peticiones de una plataforma, y se envíen los contenidos a través de la red a otro navegador o dispositivo que implementa el conjunto apropiado de protocolos en tiempo real.

Pick Contacts Intent
Esta especificación define una API que proporciona acceso a la libreta de direcciones unificada de un usuario.

Pick Media Intent
La intención de **Pick Media Intent** es definir un **Web Intent** que permita el acceso a la galería multimedia de un usuario desde dentro de una aplicación Web. Se define, tanto un par Acción/Intención que selecciona esta operación, como el formato de los datos multimedia que devuelven los servicios que implementen esta especificación.

Fullscreen
Esta especificación define una API para permitir que los elementos que se presenten a pantalla completa.

Quota Management API
API para administrar el uso y la disponibilidad de los recursos de almacenamiento local.

Selectors API Level 2
La especificación de los selectores API define métodos para recuperar los nodos Element desde el DOM, haciéndolos coincidir con un grupo de selectores (tal como se utiliza en el CSS).

Web Notifications
Este documento define un API para la visualización de una notificación al usuario.

Gamepad
Define una interfaz de bajo nivel que representa los dispositivos *gamepad*.

Pointer Lock
API que proporciona acceso a los datos de secuencia de comandos de movimiento del ratón primas, mientras que bloquean el objetivo de los eventos del ratón a un solo elemento y retirar el cursor de la vista.

URL
Esta especificación define el término URL, varios algoritmos para tratar con URLs y una API para construir, analizar y resolver URLs.

File API: Directories and System

Define una API para navegar por las jerarquías del sistema de archivos y secciones de espacio aislado del sistema local de archivos del usuario.
File API: Writer Esta especificación define una API para escribir en archivos desde las aplicaciones web.
MediaStream Capture Scenarios Este documento recopila los escenarios de destino para la API de captura multimedia que permite el acceso a las capacidades de entrada multimedia para aplicaciones Web, usando JavaScript.
Audio Processing API Esta especificación presenta y compara dos API del lado del cliente para procesar y sintetizar secuencias de audio en tiempo real en el navegador.
The Messaging API Esta especificación define una API que proporciona acceso a la funcionalidad de mensajería en el dispositivo, incluyendo SMS, MMS y correo electrónico.
Permissions for Device API Access (documento de 2010) Este documento identifica los permisos que se necesitan para el uso específico de las API del lado cliente, que otorgan acceso y permiten operaciones sobre datos sensibles.
The System Information API (documento de 2010) Esta especificación define una API para proporcionar a las aplicaciones web el acceso a varias propiedades de hardware del sistema en ejecución, incluyendo el estado de la batería y el ancho de banda de la red actual.
Web Forms 2.0 (documento de 2009) Esta especificación define la Web Forms 2.0, una extensión de las funciones de los formularios que se encuentran en el capítulo *Formularios de HTML 4* y los correspondientes interfaces DOM2 HTML. *Web Forms 2.0* se aplica tanto a los agentes de usuario HTML

como a los XHTML. Proporciona nuevos campos de entrada fuertemente tipados, nuevas interfaces DOM, nuevos eventos DOM de validación y seguimiento de dependencias, y la posibilidad de enviar e inicializar formularios en XML. También unifica y codifica las prácticas existentes en áreas que no han sido documentadas con anterioridad, y aclara algunas de las interacciones de los controles de formulario HTML y CSS.

HTML4, XHTML1.1 y el DOM se extienden así de manera que se obtiene una clara ruta de migración de formularios HTML existentes, aprovechando los conocimientos de los autores.

Silenciosamente a veces, algunas de estas API han sido ya implementadas en ciertos navegadores (incluso hace tiempo atrás), por lo que pueden resultar familiares al lector.

En este capítulo, como hemos indicado más arriba, nuestro propósito es actualizar la información y mostrar el funcionamiento de las más características o que están teniendo una mejor acogida por la comunidad y gozan ya de un nivel de implantación suficiente.

Un paso previo: el nuevo modelo de carga de *scripts*

Como ya hemos visto en capítulos anteriores y seguramente conoce el lector, la forma de programar estas API con JavaScript es mediante elementos **<*script*>** empotrados en la página o mediante ficheros separados (normalmente, de extensión *.js*, aunque esto no es imprescindible).

Hemos preferido dejar para éste último capítulo el comentario de las novedades que el elemento **<script>** presenta en esta versión en parte porque se trata de una sola novedad, y en parte porque parece más lógico diferir hasta este punto esa novedad, que no es otra que la presencia del nuevo atributo ***async*** que está diseñado para situaciones en las que, sin existir dependencias directas de un fichero concreto, se precisa que el *script* se ejecute lo antes posible o esté disponible inmediatamente después de procesar los primeros elementos de la

página.

Hasta esta versión, disponíamos de un atributo ***defer*** que permitía eso: diferir la carga del *script* para evitar llamadas bloqueantes en el proceso. Con la presencia de ***asycnc*** (nuevo en HTML 5) se aportan soluciones muy convenientes para muchos de estos escenarios.

Los hechos

Según un estudio hecho recientemente sobre los tiempos de carga de JavaScript, utilizando los primeros 100 sitios indexados en ***Alexa***, el gráfico de diferencias de la carga con y sin *scripts* ofrecía una diferencia del 31%, como podemos ver en el gráfico adjunto:

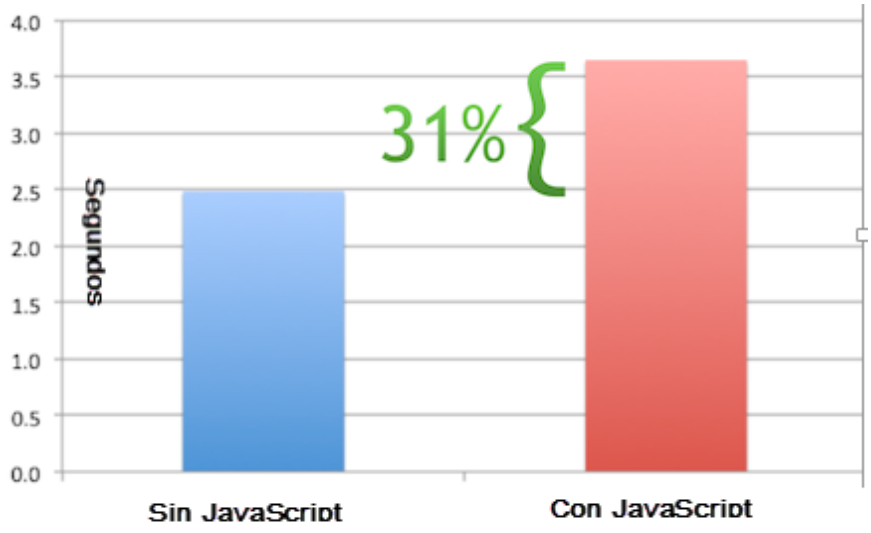


Fig. 1: Resultados (en segundos) de la carga de 100 sitios con y sin JavaScript

Por ejemplo, es muy común que tengamos en nuestros sitios referencias a *scripts* encargados de la auditoría (como *Google Analytics*), de las API de redes sociales, (como las de *FaceBook* o *Twitter*), y otras similares.

La solución propuesta

Bastaría una pequeña modificación en la etiqueta <*script*> de las referencias para que esta carga tuviese lugar de forma asíncrona, si bien

este proceso sólo funciona cuando dichos *scripts* se encuentran en ficheros separados y se produce la carga mediante un enlace aportado por la etiqueta ***<script>*** estando presente el atributo ***src*** para indicar la carga del fichero como recurso externo. Un ejemplo básico de funcionamiento, tendría el siguiente formato:

```
<head>
    <title>DanySoft</title>
    <script async src="FicheroExterno.js"></script>
</head>
```

Quizá, la situación ideal quede representada por el uso adecuado de las dos palabras reservadas, siendo defer la que podemos utilizar en caso de que no exista soporte para el proceso asíncrono en el navegador del usuario. Este atributo, lo que hace es diferir la carga del script hasta que toda la página está procesada y puede utilizarse conjuntamente con async para que sirva de alternativa, o *fallback* en muchos casos.

De esta forma, podríamos modificar la sentencia de carga de la siguiente manera:

```
<script async defer src="FicheroExterno.js"></script>
```

Con lo que, si el navegador no soporta la carga asíncrona, se opta por la opción diferida. Este segundo atributo funciona bien en la mayoría de los navegadores actuales.

Selección de las API y situación actual

De la enorme lista disponible que hemos visto en el primer apartado, hemos seleccionado unas cuantas que, por su presencia en los navegadores, su nivel de implantación actual, su aceptación por la comunidad y sus perspectivas de uso futuras, pueden implementarse ya hoy aportando funcionalidades y características nuevas y atractivas a usuarios y desarrolladores.

Puestos a hacer una división: podemos separarlas en la misma forma que

la W3C ha diseñado sus logotipos (ver figura siguiente), o por su característica funcional. En el primer caso, la división propuesta por la W3C establece divisiones más generales, que también implican al código de marcado y a CSS3:

Fig. 2: División de las tecnologías HTML5

Como vemos, las 4 primeras del gráfico no tienen que ver JavaScript (o no de forma principal), mientras que las 4 últimas sí.

- Las API vinculadas a los dispositivos y navegadores
- Las API de Almacenamiento
- Las API de Comunicaciones
- Las API de Rendimiento

Comenzamos por las propias de los dispositivos, recordando que –en muchos casos- el estándar está implementado utilizando "*vendor prefixes*" o sea objetos que llevan el prefijo del vendedor que los implementa (de forma parecida a como hemos visto en CSS3.

Las API vinculadas a los dispositivos y navegadores

Las más destacadas dentro de este apartado, son:

- La API ***FullScreen***, que permite que un elemento pueda ocupar todo el espacio disponible en la pantalla del usuario (inicialmente, se pensó para vídeo, pero posteriormente se ha extendido la especificación a otros elementos).
- La API ***Pointer Events***, definida en la W3C como un mecanismo agnóstico al dispositivo, pero capaz de capturar cualquier tipo de mecanismo de interacción del usuario con la IU.
- La API ***Drag & Drop*** (Arrastrar y Soltar), que permite programar páginas que tengan elementos móviles que el usuario puede reubicar o arrastrar elementos del sistema de archivos
- La API ***History***, que permite controlar el histórico de visitas y resulta especialmente útil en contextos de AJAX donde cambia el contenido pero no la URL de la página.
- La API ***Geolocation***, que –dependiendo de las capacidades de conexión exterior del dispositivo- aporta datos sobre la ubicación geográfica del usuario.

La API *FullScreen* (Pantalla completa)

Se trata de una de las API de implementación más reciente en los navegadores, aunque su documento oficial (en estado *Working Draft*) data de Julio/201286. No obstante, por razones diversas, no ha encontrado soporte completo hasta muy recientemente, y eso, en algunos casos como IE11, contando con que la versión es todavía una *Preview*.

Eso nos plantea inicialmente un problema añadido, que es el de la detección correcta de esta característica en nuestro código, ya que tendremos que recurrir a una rutina de comprobación múltiple para saber si existe tal soporte.

La manera en que un elemento solicita mostrarse a pantalla completa, es mediante una llamada a *requestFullScreen*. Debido al soporte de vendedor, estas llamadas se convertirán en realidad en:

[86] http://www.w3.org/TR/2012/WD-fullscreen-20120703/

- ***msRequestFullscreen***
- ***webkitRequestFullScreen*** (válido para Opera igualmente)
- ***mozRequestFullScreen***

Naturalmente, previo a la llamada debiéramos comprobar el soporte. Por otra parte, el usuario puede habilitar o deshabilitar esta característica (fuera aparte de que esté soportada), lo que requiere que comprobemos realmente si se encuentra disponible, cosa que nos permiten las versiones respectivas: ***msFullScreen***, ***webkitFullScreen*** y ***mozFullScreen***, respectivamente.

De la misma forma podemos programar (y capturar) una solicitud de salida del estado de Pantalla Completa, mediante el método (con prefijo) ***exitFullScreen***. Teniendo en cuenta que podemos determinar el estado de cualquier elemento mediante la propiedad **fullScreenElement** (una vez más con su prefijo).

El siguiente ejemplo, basado en uno de los ejemplos oficiales del MSDN, muestra una página con dos elementos DIV de colores distintos, y los programa para que se pongan a pantalla completa (o salga de ella) al pulsar sobre ellos:

```
<!DOCTYPE html>
<html>
<head>
    <title>Demo de Fullscreen API</title>
</head>
<body>
    <div id="div1" class="fullScreen" style="width:
400px; height: 250px; background-color: yellow">
        Bloque Amarillo que se pondrá a pantalla completa
    </div>
    <div id="div2" class="fullScreen" style="width:400px;
height: 250px; background-color: red">
        Bloque Rojo que se pondrá a pantalla completa
    </div>
    <script type="text/javascript">
        var enPCompleta = false;
        // Para controlar el estado fullscreen
```

```
        var elementos =
document.getElementsByClassName("fullScreen");
        for (var i = 0; i < elementos.length; i++) {
     // Programamos los eventos click sobre cada elemento
     // para que activen/desactiven la pantalla completa
            elementos[i].addEventListener("click",
function (evt) {
                if (enPCompleta == false) {
                    irAPCompleta(evt.target);
    // abrir en P.Completa
                } else {
                    salir();
                }
            }, false);
        }

        function irAPCompleta(divObj) {
            if (divObj.requestFullscreen) {
                divObj.requestFullscreen();
            }
            else if (divObj.msRequestFullscreen) {
                divObj.msRequestFullscreen();
            }
            else if (divObj.mozRequestFullScreen) {
                divObj.mozRequestFullScreen();
            }
            else if (divObj.webkitRequestFullscreen) {
                divObj.webkitRequestFullscreen();
            }
            enPCompleta = true;
            return;
        }

        function salir() {
            if (document.exitFullscreen) {
                document.exitFullscreen();
            }
            else if (document.msExitFullscreen) {
                document.msExitFullscreen();
```

```
            }
            else if (document.mozCancelFullScreen) {
                document.mozCancelFullScreen();
            }
            else if (document.webkitCancelFullScreen) {
                document.webkitCancelFullScreen();
            }
            enPCompleta = false;
            return;
        }
    </script>
</body>
</html>
```

Omitimos la captura de la salida, que es fácilmente imaginable por el lector.

La API *Pointer Events* (Eventos de dispositivo apuntador)

El estándar oficial define esta API como un conjunto funcional cuyo objetivo primordial es reducir el coste de codificación para múltiples tipos de entrada y ayudar con la ambigüedad que introducen los nuevos dispositivos apuntadores respecto a eventos de ratón.

Por eso, define una forma más abstracta de entrada, denominada puntero. Un puntero puede ser cualquier punto de contacto en la pantalla realizada por un cursor de ratón, un lápiz óptico, un toque manual (incluyendo *multi-touch*), u otro dispositivo señalador.

Este modelo hace que sea más fácil escribir páginas y aplicaciones que funcionan bien sin importar qué hardware tiene el usuario. Para los escenarios en que se desea la manipulación específica del dispositivo, esta especificación también define las propiedades para inspeccionar el tipo de dispositivo que produce el evento.

Una nota sobre el modelo de IU

Hay que tener presente, sin embargo, que al programar para este tipo de dispositivos (o una página de propósito general), debemos respetar algunos principios básicos de diseño:

- El destino de un elemento tipo "*touch*" debe tener, al menos 40 píxeles
- Debemos evitar "esconder" contenido mediante técnicas ***:hover***
- Debemos utilizar tipos de entrada específicos de HTML5
- En lo posible, debemos utilizar el "***scrolling***" y "***zooming***" nativo.
- Si queremos utilizarlo actualmente, podemos utilizar los *PolyFills* disponibles en este momento, como ***hand.js***[87] o ***pointer.js***[88].

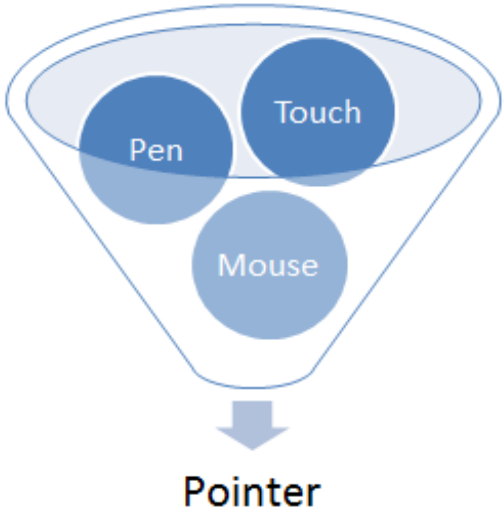


Fig. 3: Un puntero es una representación independiente del hardware, que puede orientar una coordenada (o un conjunto de ellas) en la pantalla.

En principio, su programación es sencilla, si bien hay que tener en cuenta que –de momento- los recursos están implementados

[87] http://aka.ms/handjs
[88] http://aka.ms/pointerjs

independientemente por cada vendedor, de forma que tendremos que utilizar prefijos que se corresponden con las definiciones del estándar y anteponen la misma secuencia que vimos ya en el apartado CSS (o sea, MS, *webKit*, etc.).

En concreto, el estándar define un bloque de eventos posibles que se corresponden con los que probablemente conocerá el lector para un ratón, como muestra la tabla siguiente (que indica, además, si son síncronos o no):

Tipo Evento	Sínc./Asínc.
pointerdown	**Síncrono**
pointerup	**Síncrono**
pointercancel	**Síncrono**
pointermove	**Síncrono**
pointerover	**Síncrono**
pointerout	**Síncrono**
pointerenter	**Síncrono**
pointerleave	**Síncrono**
gotpointercapture	**Asíncrono**
lostpointercapture	**Asíncrono**

Tabla 1: Tipos de eventos "pointer"

Por ejemplo, supongamos que tenemos asociado un código para dibujar un objeto ***canvas*** (llamado ***lienzo***, en el ejemplo) con un par de instrucciones:

```
lienzo.addEventListener('mousedown', comienzaDibujo);
lienzo.addEventListener('mousemove', dibujar);
```

Para cambiar esta definición a la nueva especificación y que todo funcione exactamente de la misma forma, bastaría con lo siguiente:

```
lienzo.addEventListener('MSPointerDown', comienzaDibujo);
lienzo.addEventListener('MSPointerMove', dibujar);
```

Solamente con esto, el resultado de operar con el ratón sobre nuestro lienzo, sería idéntico al anterior. No obstante, dado que este objeto (u otros sobre los que queramos actuar) tiene un comportamiento predeterminado, es preciso anular ese comportamiento para poder aprovechar realmente esta posibilidad, y eso se consigue con la propiedad **touch-action** que también define este estándar.

Esto podemos definirlo de forma distinta para distintas regiones de una página, de forma que podamos adaptar comportamientos diversos, también. Y una de las formas de implementarlo es mediante una regla CSS aplicada al elemento sobre el que se desea actuar. Por ejemplo, si queremos anular el comportamiento predeterminado de nuestro ***canvas*** podemos usar algo así:

```
<style>
    #Lienzo {
        border: 2px solid black;
        -ms-touch-action: none;
    }
</style>
```

Desafortunadamente, el único navegador que soporta esto de momento es IE en sus versiones 10 y 11 pero Mozilla lo está implementándolo igualmente, y Chrome no tardará en hacerlo.

Podemos ver perfectamente este soporte al codificar el estilo anterior en V. Studio 2013, como muestra la figura siguiente:

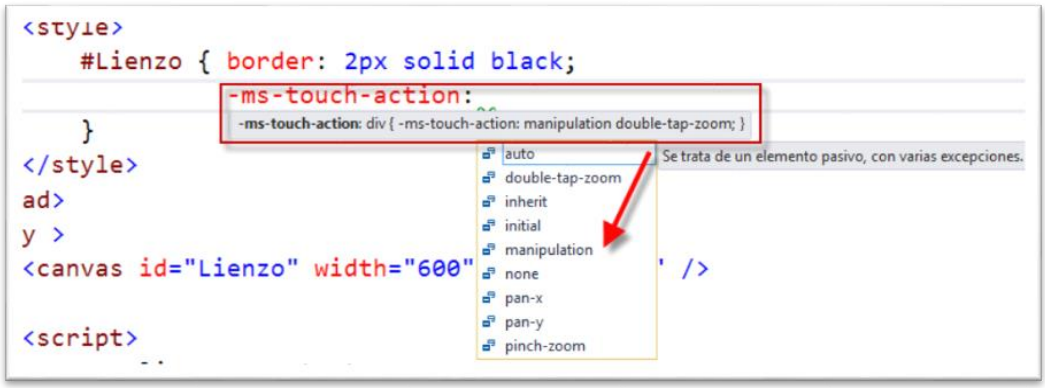


Fig. 4: Soporte de Pointer Events en Visual Studio

Una vez hecho esto, disponemos plenamente de eventos de dispositivo apuntador totalmente independientes del dispositivo y podemos probar nuestras páginas con cualquier navegador que lo soporte, o usando los PolyFills, en cualquiera de los navegadores modernos.

La implementación de Polyfills

El uso de *PolyFills* es tan sencillo como como cambiar los nombres de los eventos con prefijo por sus equivalentes del estándar (en este caso, por *PointerDown* y *PointerMove*) y añadir la librería **hand.js** a la página donde queramos utilizarlo, como si fueran eventos típicos de ratón. Eso es todo.

El lector puede comprobar este extremo (y probarlo on-line) si entra en la dirección indicada más arriba sobre ***hand.js*** y navega hacia el centro del artículo. Contiene un elemento ***canvas*** que reaccionara al paso del ratón y –si lo probamos desde una tableta o móvil- el funcionamiento será equivalente.

Naturalmente, el uso de un nuevo mecanismo de entrada como este supone consideraciones de diseño y otros "extras" como el soporte "multi-touch". Para más detalles, vea las indicaciones y sugerencias del final de este capítulo en el apartado de "Referencias".

La API *Drag & Drop* (Arrastrar y soltar)

La especificación de *Drag&Drop* es parte del bloque de HTML5 pero algunos navegadores ya lo soportaban desde tiempo atrás. De hecho, el soporte de esta funcionalidad ha sido muy variado en modo de implementación, aunque estaba presente en otros navegadores actuales. Por ejemplo, en IE9 y versiones anteriores (incluso hasta IE5), se soportaba mediante el objeto **dataTranfer** y permitía esta operación en imágenes, enlaces y texto.

De hecho, la propuesta actual, nace de la iniciativa de Microsoft, y se basa en las propuestas implementadas para las versiones anteriores de IE. Su fundamento HTML es el atributo **draggable** (aparte de las API que vamos a comentar) que puede situarse prácticamente en cualquier elemento, pero define 3 formas de comportamiento, según vemos en la tabla adjunta:

Palabra clave	Descripción
true	El contenido se puede arrastrar.
false	El contenido no se puede arrastrar.
auto	El contenido asume el comportamiento predeterminado del explorador (el texto, los vínculos y las imágenes se pueden arrastrar; otros elementos no).

Tabla 2: Valores y comportamientos asociados al atributo draggable

La especificación oficial (http://www.w3.org/TR/2010/WD-html5-20101019/dnd.html) así lo indica, aunque para más datos sobre su programación, preferimos recomendar la versión para desarrolladores de la especificación (Ver referencias, o el apartado correspondiente en MSDN).

Lo primero sería comprobar la disponibilidad de esta API mediante una sencilla comprobación inicial. Podemos utilizar *Modernizr,* hacer una simple llamada y/o establecer el *fallback* adecuado:

```
if (Modernizr.draganddrop) {
    // Hay soporte HTML5 de Drag&Drop.
} else {
    // Ofrecer una solución de fallback
}
```

Ahora, basta con entender que el proceso de arrastrar y soltar es similar al que tiene lugar en cualquier mecanismo de comunicación; esto es, existe un emisor (el usuario), un receptor (el elemento sobre el que se suelta el objeto), un canal (el contexto de ejecución que, aquí se traduce además en el objeto **dataTransfer**) y una información transmitida (la parte que nos interese del objeto inicial que el usuario selecciona).

Así que si aplicamos a los elementos de la página estas ideas, nos enfocaremos en los posibles objetos destino de la información, y el mecanismo de recepción de los datos (y lo podemos acompañar de alguna pista visual que indique al usuario el avance del proceso).

En la parte HTML5, nos basta con un elemento ***img*** y un ***div*** que albergarán una imagen y un bloque de texto cualquiera. Para hacer que sean un origen potencial de un proceso *Drag&Drop*, basta con añadirles el atributo ***draggable="true"*** (ya hemos comentado los otros valores posibles: ***false*** y ***auto***). El mecanismo de "*feedback*" visual, es normalmente aportado por el propio navegador ofreciendo una imagen atenuada del elemento durante el proceso.

En el receptor, optamos por un elemento ***div*** con un identificador, para referenciarlo posteriormente en el código JavaScript. El código, sería simplemente lo siguiente:

```
<h2>Arrastrar y Soltar</h2>
<section id="Origenes">
    <img src="/Graficos/HTML5_Logo.png" draggable="true"
id="grafico" />
    <div id="divTexto" draggable="true">Texto
desplazable</div>
</section>
<!-- Zona de destino -->
<div id="CajaDestino"></div>
```

En la parte de JavaScript, deberemos implementar un mecanismo de escucha para que cuando se produzca el inicio de un proceso de este tipo se almacene la información deseada (eso lo decide el desarrollador) sobre el objeto a transferir. Esto se hará mediante un manejador del evento ***dragstart***.

Por último, cuando se produce el evento ***drop***, tendremos que leer del objeto ***dataTransfer*** la información almacenada en él y operar en consecuencia (actualizar la IU, en este caso).

```
<script>
    // Primer modelo de Drag&drop (arrastre de
    // elementos en la propia página

    /* Arrastre de 2 elementos distintos */
    window.addEventListener('load', iniciarDD, false);

    function iniciarDD() {
        // Configuración de diversos orígenes de arrastre
        origen1 = document.getElementById('grafico');
        origen1.addEventListener('dragstart',
evArrastrado, false);

        origen2 = document.getElementById('divTexto');
        origen2.addEventListener('dragstart',
evArrastrado, false);

        destino = document.getElementById('CajaDestino');
        destino.addEventListener('dragenter', function
(e) {
            e.preventDefault();
        }, false);
        destino.addEventListener('dragover', function (e)
{
            e.preventDefault();
        }, false);
        destino.addEventListener('drop', evSoltado,
false);
    }
```

```
    function evArrastrado(e) {
        if (e.srcElement.id == "grafico") {
            var codigo = '<img src="' +
e.srcElement.getAttribute('src') + '">';
            e.dataTransfer.setData('Text', codigo);
        } else {
            e.dataTransfer.setData('Text',
e.srcElement.innerHTML);
        }
    }

    function evSoltado(e) {
        e.preventDefault();
        destino.innerHTML =
e.dataTransfer.getData('Text');
    }
</script>
```

Nótese que en el caso del gráfico, lo que almacenamos es una cadena que expresa un nuevo elemento ***img*** con su referencia a la imagen correspondiente, mientras que en el caso del ***div*** nos conformamos con contenido del elemento (el texto).

Por supuesto, la salida gráfica es la que cabe esperar, con la particularidad de que no tenemos que programar ninguna acción para que el navegador aporte las pistas visuales durante el proceso que indican al usuario si es posible arrastrar y/o soltar un elemento en una cierta zona de la página, como vemos en la figura siguiente, que es la salida del código anterior:

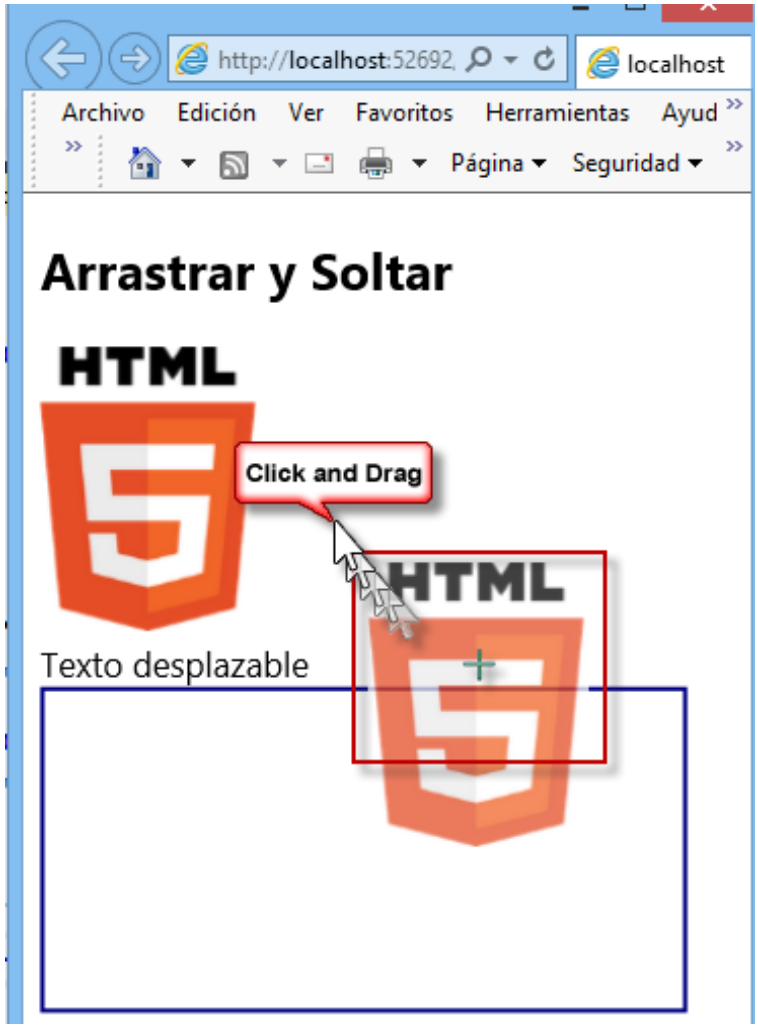


Fig. 5: Proceso Drag&Drop e indicadores visuales

En ambos casos, el manejador del evento ***drop*** asignará al receptor un contenido que es una cadena, pero que será interpretada como texto o como un nuevo elemento del DOM, actualizando la IU.

Drag&Drop desde archivos del sistema

Además de la posibilidad de mover elementos de la misma página, Microsoft, añadió desde IE10 el atributo ***files*** al objeto ***dataTransfer*** lo que permite recoger en un *array* los archivos seleccionados por el

usuario.

En este caso, el código es muy similar, con la diferencia de que, en el manejador del evento ***drop***, recorremos el array de archivos almacenado en ***dataTransfer***, y –utilizando el API de archivos, de la que hablamos más adelante en este capítulo- mostramos la información de éstos en una lista del elemento de destino. El código sería el siguiente:

```
// Arrastrar archivos del sistema local
// Al terminar la carga inicial, lanzará la funcion
iniciar
window.addEventListener('load', iniciar, false);

function iniciar() {
    destino = document.getElementById('CajaDestino');
    destino.addEventListener('dragenter', function (e) {
        e.preventDefault();
    }, false);
    destino.addEventListener('dragover', function (e) {
        e.preventDefault();
    }, false);
    destino.addEventListener('drop', soltado, false);
}

function soltado(evento) {
    evento.preventDefault();
    var archivos = evento.dataTransfer.files;
    var lista = '';
    for (var f = 0; f < archivos.length; f++) {
        lista += 'Archivo: ' + archivos[f].name + '  -
Tamaño: ' +
            archivos[f].size + ' bytes' + '<br>';
    }
    destino.innerHTML = lista;
}
```

Advierta que, en ambas situaciones, los eventos **dragover** y **dragenter** usan ejecutan el método ***preventDefault()*** para evitar la administración y propagación automática del evento, lo que podría provocar resultados

impredecibles.

Omitimos la salida gráfica, prácticamente idéntica al a anterior, excepto en que muestra los nombres de los archivos arrastrados y su tamaño.

Esta técnica simplifica la creación de programas, como un cliente de correo electrónico en el cual puedes arrastrar datos adjuntos a un mensaje o agregar fotos a una página de galería.

El API *History* (Historial de Navegación)

La especificación oficial[89] de esta API se describe en la sección 5.4.2 de la especificación HTML5, e incluye métodos para administrar la pila de historial y la dirección URL de un sitio. Ofrece a los usuarios finales una experiencia natural a la hora de utilizar los botones Atrás y Adelante del navegador y una mejora del rendimiento en las actualizaciones de poca carga para páginas sin navegación ni cargas completas.

Su nivel de soporte es completo en todos los navegadores modernos como podemos ver en el cuadro de soporte que nos aporta el sitio especializado **CanIUse**[90].

Se basa en un objeto del mismo nombre (***history***), vinculado a ***window,*** que posee algunos métodos útiles para el control de la pila del historial de navegación, como ***pushState*** y ***replaceState***. Concretamente, este objeto aporta las siguientes funcionalidades:

- **history.length**
  - Devuelve el número de entradas en el historial de sesión.
- **history.state**
  - Devuelve el objeto ***state*** actual.
- **history.go([*delta*])**
  - Retrocede o avanza el número especificado de pasos en el historial de sesión.

[89] http://www.w3.org/TR/html5/browsers.html#the-history-interface

[90] http://caniuse.com/#search=History

    - Un valor [delta] de cero volverá a cargar la página actual.
    - Si [delta] está fuera de rango, no hace nada.
- **history.back()**
    - Retrocede un paso en el historial de la sesión.
    - Si no hay una página anterior, no hace nada.
- **history.forward()**
    - Avanza un paso en el historial de sesión.
    - Si no hay una página anterior, no hace nada.
- **history.pushState(*data*, *title* [, *url*])**
    - Inserta los datos pasados en el historial de sesión, con el título pasado y –si se incluye- la URL indicada.
- **history.replaceState(*data*, *title* [, *url*])**
    - Actualiza el registro actual en el historial de sesión con los valores pasados como argumentos.

De forma que con los dos últimos métodos, ***pushState()*** puede crear una nueva entrada en el historial y, opcionalmente, incluir un objeto de estado, y usando ***replaceState()***, es posible modificar el elemento activo del historial.

El significado exacto de los 3 parámetros es el siguiente:

- ***data***: es un objeto que representa el estado que se quiere asociar a una nueva entrada del historial. Este objeto se devuelve como la propiedad ***state*** del evento ***popState***.
- ***title:*** es un título que puede pasarse opcionalmente (Actualmente, Internet Explorer 10 lo omite).
- ***url:*** dirección URL relativa o absoluta en el mismo dominio u origen de la dirección URL actual

Como ya es habitual, el soporte de esta API en Visual Studio 2012/2013 es completo, como podemos ver en la siguiente captura del editor:

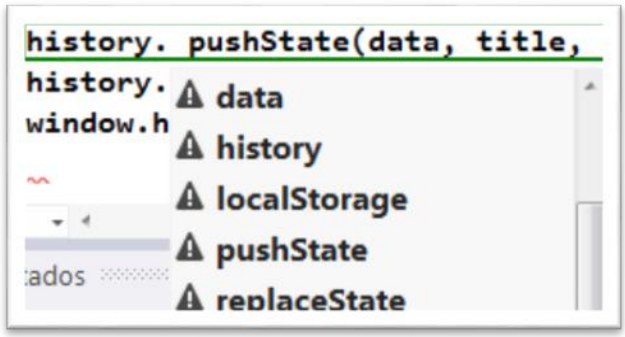


Fig. 6: Soporte del API History en V. Studio 2012/2013

Con estos nuevos métodos se puede modificar la ruta o el *hash* (la información que va después de la almohadilla en la dirección URL), o el segmento de consulta que sigue al signo de interrogación (?) de la dirección URL pero, por seguridad, no es posible modificar el dominio o el origen de la dirección URL actual.

El evento window.onpopstate

Este evento notifica a la aplicación cuando el usuario navega entre dos entradas del historial, siempre que, al menos una, haya sido creada con **history.pushState()** o **history.replaceState()**. La notificación se produce cuando el usuario hace clic en los botones Atrás o Adelante, o cuando se llama a cualquiera de los dos métodos.

La forma de diferenciar entre los dos en el controlador de eventos es mediante una sintaxis como la siguiente:

```
window.onpopstate = function (event) {
    if (event.state) {
        // History ha cambiado mediante
        // pushState o replaceState.
        alert(event.state);
    } else {
        // Carga normal de página
    }
}
```

Naturalmente, el parámetro del evento **popState** contiene el estado **pushState** o **replaceState** (que habrá sido establecido por el parámetro de datos). Y en caso de que se cargue una página, el estado será sin definir.

Finalmente, tenga en cuenta que la propiedad **history.state** también contiene el objeto de estado y que cuando una página se carga inicialmente, ese estado será NULL. Cuando se llama a ***pushState***, la pila de avance se borra, pero no sucede así al llamar a **replaceState.**

El API *Geolocation* (Geo-Localización)

Otra de las API más populares de la Web es la Geo-Localización[91]. Una API que ya ha alcanzado el nivel de ***Proposed Recommendation*** y que consiste en la capacidad de un sitio o aplicación web para determinar la procedencia geográfica del visitante (o la ubicación del usuario), pudiendo responder adecuadamente a esa circunstancia, personalizando la interfaz, o –simplemente- ayudando a llevar una auditoría por zonas de cara a los administradores. Los posibles usos, además, se multiplican con la informática móvil, ya que puede proveernos de mapas asociados, recursos disponibles en una zona, etc.

El procedimiento no es complejo, y esencialmente, se basa en un servicio al que se consulta para obtener los datos de ubicación. Esos datos son devueltos en términos de latitud y longitud geográficas, (junto a un conjunto adicional de metadatos) y, a su vez, convertidos -si es necesario- en una posición dentro de un mapa explicativo.

Tanto latitud como longitud pueden ser expresadas en forma decimal (Lat. 42,123122) o en formato de sexagesimal (Lat. 42º 23' 14"), si bien cuando utilizamos el API de geo-localización, ésta nos devuelve la información en formato decimal.

[91] La especificación oficial del estándar se encuentra en http://www.w3.org/TR/geolocation-API/

¿De dónde proviene la información?

Este es el punto clave a determinar y depende, en parte, del dispositivo que estemos utilizando. El API simplemente expone un conjunto de mecanismos de acceso a la información, pero no garantiza la fiabilidad más allá de un punto. De hecho, un dispositivo puede usar cualquier de los métodos siguientes para acceder a esa información:

- Dirección IP
- Triangulación de coordenadas
    - Sistema de posicionamiento global (GPS)
    - Wi-Fi con direcciones MAC desde RFID, Wi-Fi y Bluetooth
    - Identificadores telefónicos tipo GSM o CDMA
- Definida por el usuario

El tercer caso no lo vamos a comentar aquí, pero en los dos primeros la precisión depende de muchas circunstancias, por lo que debe de tenerse esto en cuenta al elegir el método. En algunos casos, se puede optar por una combinación de los anteriores, con el objeto de ganar exactitud. Vamos a comentar someramente las distintas posibilidades.

Dirección IP

Era la única disponible hace años. Sin embargo, las ubicaciones devueltas no son de lo más preciso, ya que dependen del proveedor de servicios de Internet, y esto puede estar localizado a docenas de kilómetros de nuestra ubicación real.

La ventaja es que está disponible en cualquier momento y se procesa en el servidor, pero puede ser una operación costosa, a la par que poco fiable.

GPS

Con la única condición de que se vea el cielo desde el punto de la petición, el GPS puede ofrecer los resultados más ajustados. El problema aquí es que el proceso puede llegar a ser muy lento, lo que le hace un candidato obligado a una llamada asíncrona.

Otro problema es que no funciona adecuadamente en los espacios cerrados. Y que puede requerir de hardware adicional.

Wi-Fi

Se basa en la triangulación de la posición basándose en la distancia a 3 puntos Wi-Fi conocidos, especialmente en áreas urbanas. Funciona muy bien en espacios cerrados (al igual que abiertos), y puede obtener los datos de posición de forma muy rápida y sencilla, teniendo como único inconveniente que puede no funcionar bien cuando nos encontramos en áreas no urbanas donde la localización de puntos de conexión es esporádica.

Geo-localización desde dispositivos móviles.

Casi todos los móviles de hoy en día vienen equipados con sistemas que permiten la geo-localización por GPS. El funcionamiento se basa calculando la distancia del usuario a distintos repetidores de telefonía móvil.

Ofrece una precisión bastante aceptable, y, a veces, se combina con otros métodos de detección vía Wi-Fi.

Adolece de problemas similares a los de la localización por Wi-Fi cuando estamos en áreas rurales con menos repetidores disponibles.

Definida por el usuario

En algunos escenarios, tiene sentido que permita al usuario establecer su propia ubicación, que puede ser indicada mediante un dato urbano o los parámetros de latitud o longitud igualmente, y utilizar un servicio para ofrecer al usuario más datos relativos a su enclave actual.

Permite que un usuario conozca datos de ubicaciones distintas a la suya y en ocasiones, la introducción manual puede ser más rápida que la petición, aunque la precisión siempre dependerá de la ofrecida por el servicio.

Implementación en un par de casos prácticos

Afortunadamente, la geo-localización fue una de las primeras características que todos los navegadores que analizamos se apresuraron a implementar. Además, existen librerías especializadas que cumplen con esta función. Una de las más populares es *Gears*, que ofrece soporte en IE, Chrome, y FireFox, al menos.

Privacidad

Naturalmente, el estándar recomienda que estas opciones solo estén disponibles cuando el usuario permita el acceso. Esto forma parte de las políticas de privacidad, y los navegadores disponen de un mecanismo de protección activado de forma predeterminada. Simplemente, cuando accedemos a una página que solicita este tipo de información, el mecanismo se manifiesta mediante algún tipo de aviso.

Soporte nativo

Naturalmente, el caso ideal es aquel en que no necesitamos soporte de librerías adicionales porque lo obtenemos directamente del propio navegador. Esa es la propuesta del estándar y es lo que queremos ilustrar aquí.

Los datos geográficos en este caso se obtienen directamente del DOM, accediendo al objeto ***navigator.geolocation***, que, además de devolver dicha información, nos servirá para controlar el propio hecho del soporte, ya que una llamada a este objeto desde JavaScript, nos devolverá un valor falso en caso contrario.

Podemos implementar una funcionalidad básica haciendo un par de llamadas a estas API y reflejando el resultado en pantalla, como se muestra en el código siguiente:

```
<head>
    <meta charset="utf-8" />
    <title>Geo-localización nativa con HTML 5</title>
</head>
<body onload="comprobarNavegador()">
```

```
    <h1>Ejemplo de Geo-localización nativa con HTML
5</h1>
    <p id="nivelSoporte">La Geo-localización nativa NO
está soportada en este navegador.</p>
    <h2>Ubicación actual:</h2>
    <h5>Latitud: <span id="latitud">n/c</span></h5>
    <h5>Longitud: <span id="longitud">n/c</span></h5>
    <h5>Precisión: <span id="precision">n/c</span></h5>

<script type="text/javascript">
  function leerGeo() {
    // Función de geolozalización que hace una llamada
    // asíncrona al servidor. La función de callback
    // recoge los datos y actualiza la IU
    navigator.geolocation.getCurrentPosition(
                actualizaIU, errorEnConsulta);
}

// El objeto position contiene toda la información
function actualizaIU(position) {

    var lat = position.coords.latitude;
    var lon = position.coords.longitude;
    var pre = position.coords.accuracy;

    if (!lat || !lon) {
        document.getElementById("nivelSoporte").innerHTML
=
          "Geo-Localizacion HTML5 soportada, pero no
disponible.";
        return;
    }
    document.getElementById("latitud").innerHTML = lat;
    document.getElementById("longitud").innerHTML = lon;
    document.getElementById("precision").innerHTML = pre
+ " ms.";
}

function errorEnConsulta(err) {
    switch (err.code) {
```

```
        case PositionError.PERMISSION_DENIED:
            alert("Permiso denegado por el usuario");
            break;

        case PositionError.PERMISSION_UNAVAILABLE:
            alert("Datos de ubicación no disponibles");
            break;

        case PositionError.TIMEOUT:
            alert("Petición fuera de tiempo");
            break;

        default:
            alert("Error desconocido");
            break;
    }
}
</script>
</body>
</html>
```

Hay que tener en cuenta aquí que el objeto clave que nos devuelve la información pertinente es ***position***, quien, en realidad, devuelve más información de la que estamos utilizando aquí. En concreto este objeto contiene las siguientes propiedades:

- **latitude** **(Latitud)**
- **longitude** **(Longitud)**
- **accuracy** **(Precisión)**
- **altitude** **(Altitud)**
- **altitudeAccuracy** **(Precisión de la altitud)**
- **heading** **(Dirección del movimiento, respecto al norte geográfico)**
- **speed** **(Velocidad del movimiento, en metros por segundo)**

En primer lugar, el soporte será diferente según el navegador que

utilicemos para probar este código, como ya hemos indicado, existiendo la posibilidad de que uno de ellos soporte la comprobación pero no la llamada. Esperemos que en poco tiempo se produzca una unificación de estos soportes.

En segundo lugar, todos sin excepción, nos pedirán permiso para realizar la operación indicando que si queremos permitir que otros conozcan nuestra ubicación y todos ellos guardan, si así lo queremos, la opción elegida (aceptarlo o rechazarlo). Una vez aceptada esa opción, debiéramos de obtener una pantalla similar a la de la figura 5 (esta vez hacemos la prueba en IE9, pero en IE10/IE11 es idéntica):

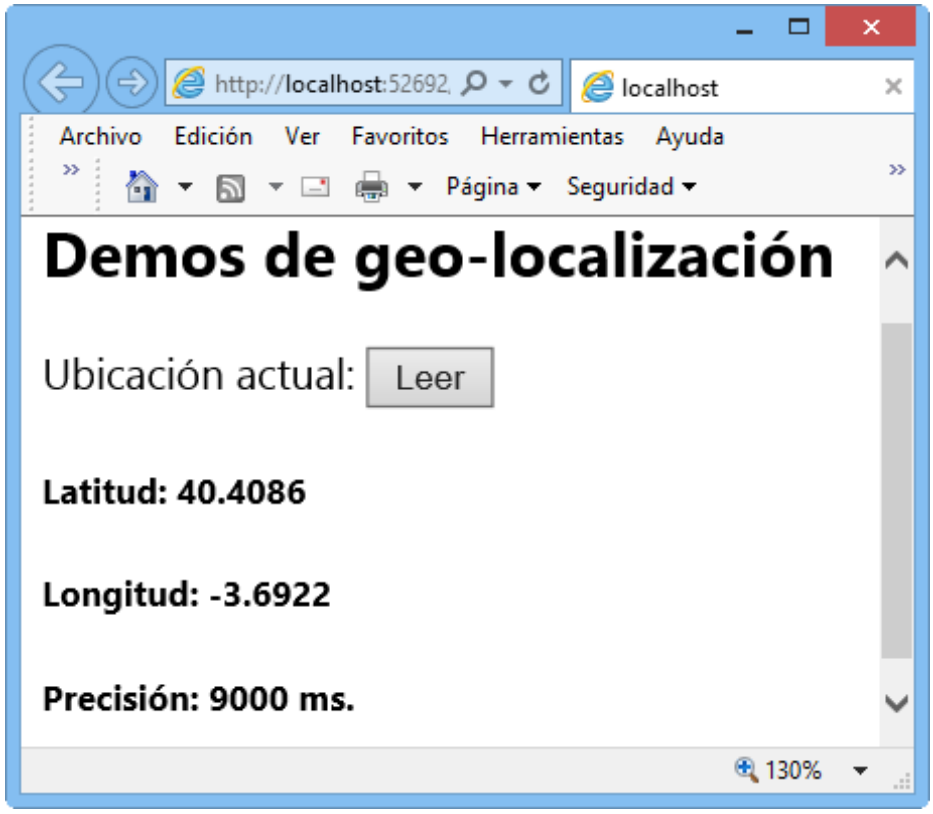


Fig.7: Salida del código anterior en IE9/IE10/IE11.

Gestión de errores

Además, siempre que realizamos una petición a un servidor (y máxime a un servicio tan especializado), es posible que obtengamos errores. La función de *callbak* **errorEnConsulta** nos permite capturar el posible error, que nos será devuelto en la propiedad **PositionError** (en el código controlamos 3 de los posibles errores devueltos).

Tenga en cuenta, no obstante, que esto es para el caso en que estemos usando la opción de navegador. Cualquiera de las otras mencionadas, deberá de seguir sus propios criterios de error.

Otras Librerías

Como ya apuntábamos al principio, la otra opción se basa en utilizar librerías especializadas, como *Gears*, pero, en este caso, no estamos hablando de API nativas del estándar sino de soluciones de terceros.

De la misma forma, resulta muy sencillo solicitar un servicio básico (o personalizado) de mapas utilizando las API disponibles para Bing Maps o Google Maps.

Obtención de mapa simple con Google Maps

Por ejemplo, si conocemos la ubicación de un sitio y simplemente queremos completar los datos con un mapa de la zona, una sencilla petición a Google Maps, nos permite hacerlo (el usuario puede obtener una clave para utilizarlo por encima del número de veces diarias que se ofrece como cortesía):

```
<!DOCTYPE html>
<html>
<head>
    <title>Mapa Sencillo</title>
    <meta name="viewport" content="initial-scale=1.0,
user-scalable=no">
    <meta charset="utf-8">
    <style>
        html, body, #map-canvas {
            margin: 0;
```

```
            padding: 0;
            height: 100%;
        }
    </style>
    <script
src="https://maps.googleapis.com/maps/api/js?v=3.exp&sens
or=false"></script>
    <script>
        var map;
        function initialize() {
            var mapOptions = {
                zoom: 8,
                center: new google.maps.LatLng(40.416775,
-3.70379),
                mapTypeId: google.maps.MapTypeId.ROADMAP
            };
            map = new
google.maps.Map(document.getElementById('map-canvas'),
                mapOptions);
        }

        google.maps.event.addDomListener(window, 'load',
initialize);

    </script>
</head>
<body>
    <div id="map-canvas"></div>
</body>
</html>
```

Este sencillo código utiliza los datos obtenidos anteriormente sobre la ubicación y obtiene el mapa que se muestra en la figura siguiente:

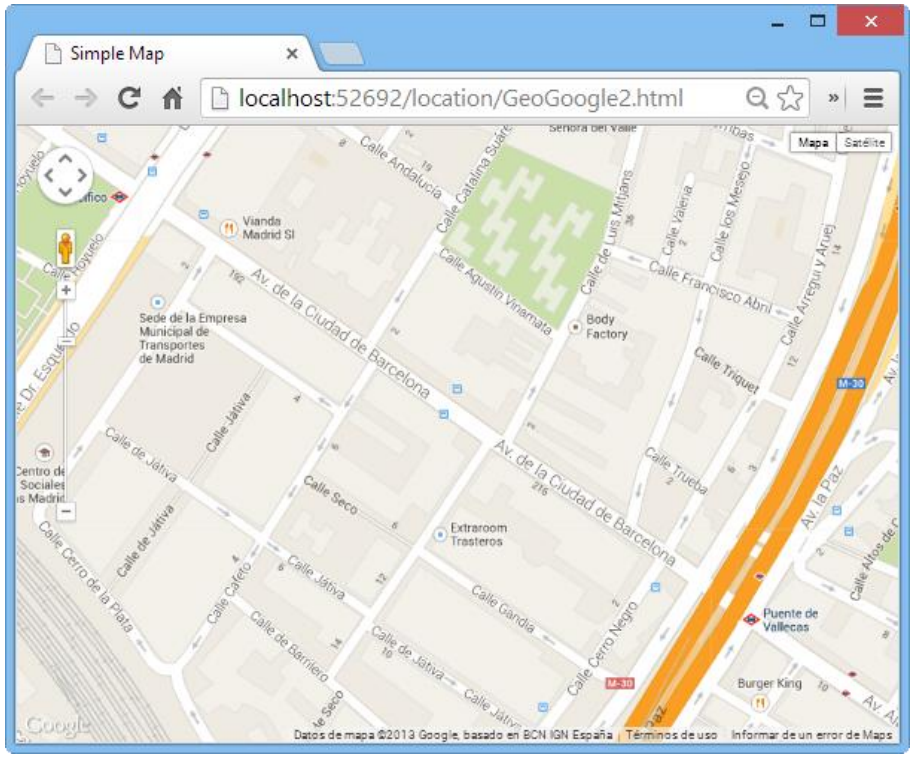


Fig. 8: Mapa básico de Google Maps sobre unas coordenadas fijas

Fusionando las dos opciones: Geo-localización y Bing Maps

En el caso de **Bing Maps**, se ofrecen servicios muy precisos de geo-localización (basta con entrar en el sitio http://bing.com/maps para advertir que la página nos ubica automáticamente en el punto del mapa más cerca de donde nos encontramos físicamente, hecha la salvedad de la precisión en estos sistemas que no se basan en GPS).

Con ese patrón de código resulta fácil implementar una puesta en marcha básica del API de Geo-localización y hacer que se nos muestre un mapa con la ubicación correspondiente.

Para ello, podemos utilizar el primero de los códigos de acceso, y sustituir el contenido de la función ***actualizaIU*** por el siguiente código JavaScript:

```
// Previamente, en la cabecera, hemos incluido una
petición al control
```

```
// Bing Maps mediante el siguiente script:

    <script charset="UTF-8" type="text/javascript"
src="http://ecn.dev.virtualearth.net/mapcontrol/mapcontro
l.ashx?v=7.0" ></script>

// El objeto position contiene toda la información
function actualizaIU(position) {
    // Crea un mapa de Bing
    _map = new
Microsoft.Maps.Map(document.getElementById("mapa"),
        {
            credentials:
"Ak3Sjc32qVKfI0CiNObHGFEuq_fIeN4kq2zzjNDEqbu9Gmjpbj7bN3wk
mCZvas8C"
        });
    // Aplica la posición al mapa
var location = new Microsoft.Maps.Location(
position.coords.latitude,position.coords.longitude);
    _map.setView({ zoom: 21, center: location });
    // Añade un marcador indicando la posición en el mapa
    var pin = new Microsoft.Maps.Pushpin(location);
    _map.entities.push(pin);
}
```

Observe el lector que realizamos muy pocos cambios. En lugar de mostrar los valores de latitud y longitud, se los pasamos al método ***Location*** del objeto ***Microsoft.Maps.*** El valor devuelto es fundamental para el funcionamiento posterior, ya que se lo pasaremos al método **setView** del objeto **Map** creado al principio, y lo volveremos a pasar al método **Pushpin** del objeto **Maps** para obtener los elementos visuales complementarios al mapa.

El objeto **Microsoft.Maps** está disponible desde el momento en que la librería ***mapcontrol*** ha sido descargada. La salida del código fuente anterior con las modificaciones hechas en esta última versión es idéntica en todos los navegadores (Ver figura siguiente):

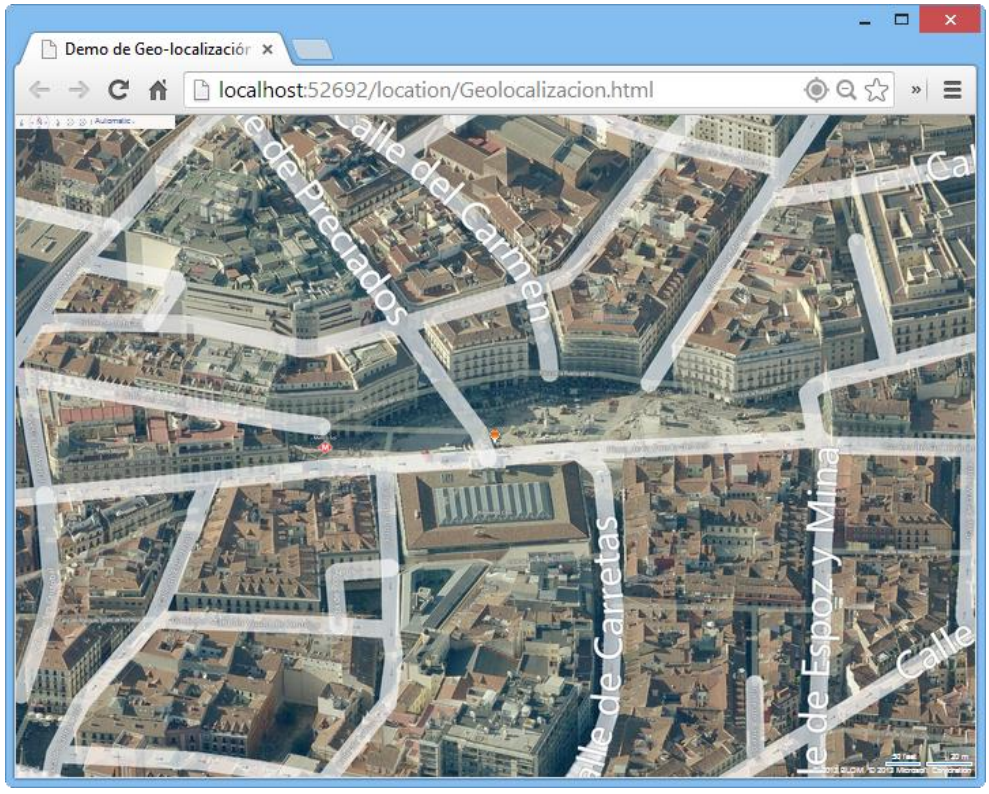


Fig. 9: Salida del código anterior

De todas, formas, tenga en cuenta que lo más recomendable en este tipo de situaciones son las llamadas asíncronas, que el estándar provee mediante el objeto ***PositionCallBack*** y ***PositionErrorCallBack***, para permitir una experiencia de usuario flexible. Además, el sistema contempla la posibilidad de establecer acciones basadas en una llamada o en varias, siendo esta última más fiable, a costa de una penalización (controlable), del rendimiento.

Las API de Almacenamiento

Las más destacadas dentro de este apartado, son:

- La API ***WebStorage***, que permite sustituir las "cookies", permitiendo al mismo tiempo una mayor flexibilidad ya que se divide realmente en dos versiones: ***localStorage*** (almacenamiento local permanente) y ***sessionStorage*** (almacenamiento local solo para la sesión activa).
- La API ***AppCache*** (Caché de aplicaciones) que permite definir sin ambigüedad que elementos deben situarse en una zona especial de almacenamiento para evitar recargas (o nuevas peticiones) desde el servidor, al volver a una página recientemente visitada.
- La API ***IndexedDB***, que permite emular (con algunas limitaciones) el comportamiento de una base de datos local.
- La API ***File***, que permite acceder (con ciertas restricciones) al sistema local de archivos.

Las API de almacenamiento local y de sesión (localStorage sessionStorage)

El objetivo de estas API es la persistencia de información entre distintas peticiones Web. La idea es que se puedan almacenar datos entre peticiones, de una forma distinta y más adecuada a la que se hace ahora mediante ***cookies.*** Esto enriquece las posibilidades de la aplicación y reduce el tráfico de red.

El soporte es completo en todos los navegadores analizados. Respecto a su operativa, estos servicios son accedidos mediante objetos vinculados con el objeto ***window: window.sessionStorage*** y ***window.localStorage.***

Se utiliza una base de datos manejada internamente por el navegador y controlada mediante estas API. Por tanto, el primer paso es la comprobación del soporte por parte del navegador utilizado. Para ello, basta con verificar si una referencia a estos objetos devuelve un valor verdadero, tal y como vemos en el siguiente código:

```
function comprobarSoporte() {
    //sessionStorage
    if (window.sessionStorage) {
```

```
        alert('Este navegador soporta sessionStorage');
    } else {
        alert('Este navegador NO soporta
sessionStorage');
    }
    //localStorage
    if (window.localStorage) {
        alert('Este navegador soporta localStorage');
    } else {
        alert('Este navegador NO soporta localStorage');
    }
}
```

Podemos llamarlo al entrar en la página que necesite estos servicios y comprobaremos esta situación en los diversos navegadores, como podemos observar en la figura 6:

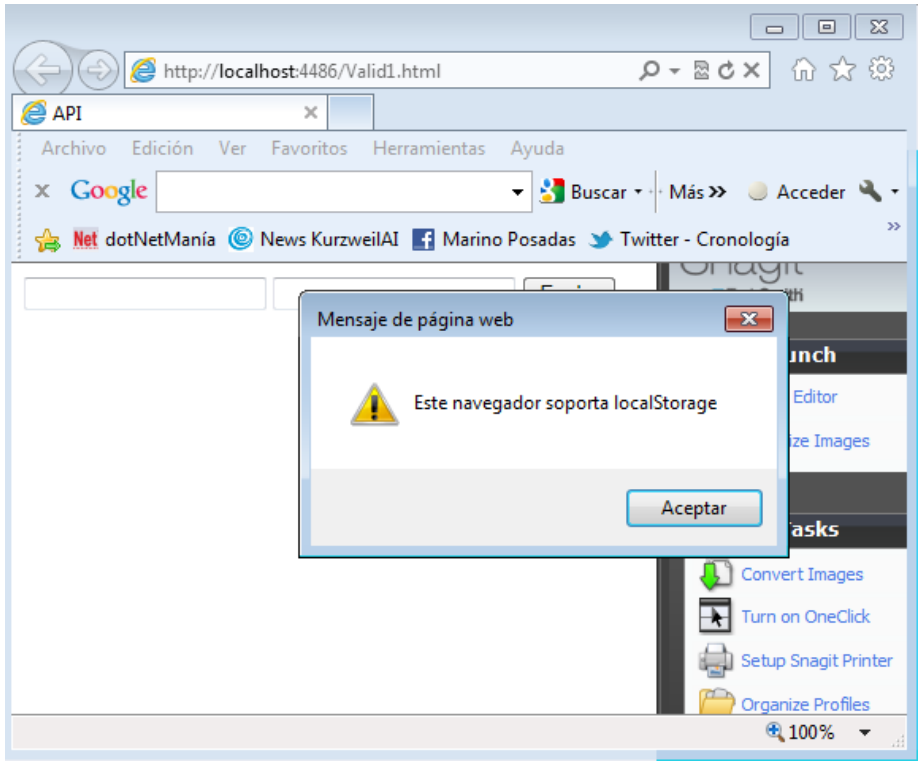


Fig. 10: Salida del código anterior en IE9/10/11

Almacenamiento y recuperación de datos de sesiones

El objeto ***sessionStorage*** dispone de varios métodos propios para asignación (escritura) y recuperación (lectura) de información. No obstante, -y por el momento- aunque las API están pensadas para soportar diversos tipos de datos, en la actualidad solo encontramos soporte para cadenas, así que las pruebas que hacemos se limitan a ellas.

Podemos implementar un sencillo sistema de lectura y escritura en una página, que nos permita grabar y recuperar un dato y mostrarlo en algún elemento de la interfaz de usuario. La instrucción de escritura utilizada es ***setItem(key, value)***, que opera recibiendo parejas clave/valor y almacenándolas para recuperación posterior en la base de datos asignada al navegador.

Esa recuperación se realiza mediante el método ***getItem(clave)*** que permite leer cualquier valor guardado a partir de su clave. Usamos una interfaz de usuario sencilla con dos botones "*Grabar*" y "*Leer*", y una caja de texto para almacenar los datos de entrada y mostrar la salida.

```
<body>
    <form action="Valid1.html" id="Formulario" >
        <p>Almacenamiento de valores de sesión</p>
        <input type="text" id="clave1" />
<input type="button" value="Grabar"
onclick="escribirClave('Dato');" /><br />
        <input type="text" id="claveLeida" />
<input type="button" value="Leer"
onclick="leerClave('Dato');" />
    </form>
</body>
```

Y en la parte de JavaScript, habilitamos dos funciones (para la lectura y escritura), que recogen el valor del elemento "*clave1*" y lo graban en la base de datos, y, vinculamos al segundo botón la lectura del valor y su asignación a la caja de texto "*claveLeida*":

```
function escribirClave(datos) {
    var valor =
document.getElementById("clave1").value;
    window.sessionStorage.setItem(datos, valor);
}

function leerClave(datos) {
    var valor = window.sessionStorage.getItem(datos);
    document.getElementById("claveLeida").value =
valor;
}
```

Todavía puede simplificarse la sintaxis aún más, utilizando las llamadas "propiedades de expansión" (*expando-properties*) para referirnos a la clave queremos asignar como si ya fuese un valor existente en la base de datos, donde sustituimos:

```
window.sessionStorage.setItem(datos, valor);
```

por

```
window.sessionStorage.clave1 = valor;
```

Y lo mismo podemos hacer en el proceso de lectura. Los ejemplos anteriores funcionan correctamente en las versiones IE9 e IE10.

Esta persistencia tiene un tiempo de caducidad que debemos considerar. Cada vez que cierre la página (o la solapa que contenga una página con este mecanismo), la información se perderá.

Por tanto, resulta un mecanismo adecuado para procesos de vida corta, como la información que debe persistirse relativa a Asistentes o a cajas de diálogo. Al igual que otros datos propios de la actividad de un usuario concreto. Téngase en cuenta que un cambio de página dentro del mismo sitio no produce la pérdida de la información. Por lo tanto, los datos guardados en formularios previamente rellenados o producto de otros procesos pueden almacenarse igualmente de esta forma.

El API de Almacenamiento local (*LocalStorage*)

Para los casos en los que es necesario que la información persistida dure más allá de la sesión o del cierre del navegador, la opción adecuada es el almacenamiento local (***localStorage***).

La implementación de esta API se ha hecho de forma totalmente coherente con la anterior, de manera que, tanto la sintaxis como la metodología de uso resultan idénticas a los ejemplos anteriores, cambiando solamente las referencias al objeto ***sessionStorage*** por ***localStorage.***

En resumen, unas API que sustituyen ventajosamente a las *cookies*, y permiten cubrir de manera muy sencilla las necesidades de almacenamiento locales vinculadas con aspectos del uso de páginas y aplicaciones.

Finalmente, una comparativa entre ambas revela las diferencias más importantes:

sessionStorage	localStorage
La persistencia se limita a la página o solapa de navegación dentro del sitio	La persistencia se mantiene incluso después de cerrar el navegador
Los valores almacenados o recuperados solo son visibles desde la página o solapa que los creó.	Los valores se mantienen entre distintas ventanas o solapas ejecutando el mismo origen URL.

Tabla 3: Comparativa de funcionamiento entre las API sessionStorage *y* localStorage

También existen limitaciones a la cantidad de información que puede guardarse en el apartado ***valor*** vinculado con una clave dada. Para establecer esto, existen unos valores de ***quota***, que es configurable por el usuario, o bien por el navegador con el permiso de éste.

No obstante, este sistema de almacenamiento de pares, no resuelve todas las necesidades posibles de cara a las aplicaciones, por lo que los

estándares promovieron otras soluciones alternativas.

Las más importantes fueron ***AppCache***, ***IndexDB*** y ***WebSQL***, si bien ésta última, por razones que no vamos a comentar aquí, ha sido abandonada y no formará finalmente parte del estándar. Todo lo contrario ha sucedido con las dos primeras que ya han pasado todos los test y se encuentran en fases avanzadas de terminación.

El API de caché de aplicaciones (*AppCache*)

Otra de las API importantes en el área del almacenamiento de información es la API ***AppCache***, que permite almacenar información relativa a las aplicaciones Web de forma local, evitando de esa forma peticiones innecesarias al servidor. Resumiendo sus principales características, podemos decir que se basa en los siguientes aspectos:

- Funcionamiento declarativo, basado en un *manifiesto* que habilita escenarios "***off-line***".
- Selección de ficheros a almacenar en la caché. Pueden ser de tipo HTML, CSS, JavaScript o recursos.
- Mejora de la disponibilidad de los recursos más allá del caché local de HTTP.
- Re-sincronía de los archivos mediante una actualización del *manifest*.

Una de las aplicaciones más interesantes de esta API es evitar que el funcionamiento de una aplicación deje de estar operativo debido a una falta de disponibilidad del servidor. Incluso en algunos casos, podríamos situar en caché todos los archivos necesarios para el funcionamiento y eliminar cualquier otro tipo de tráfico, salvo cuando fuera necesaria una actualización (si es que tal caso puede darse en la aplicación), ya que en otros casos podríamos continuar todo el proceso de forma desconectada sin perjuicio alguno.

Por ejemplo, un juego on-line podría descargarse en su totalidad y no

necesitar del concurso del servidor más que cuando este pasa de un nivel a otro (si la carga de cada nivel es muy pesada), o cuando el usuario quisiera subir sus resultados al servidor para compararlos con los de otros jugadores. En resumen, las ventajas serán principalmente tres:

- Posibilidad de navegación "off-line" por pérdida de la conexión
- Recuperación instantánea de cualquier recurso ya visitado (y situado en la caché)
- Reducción considerable de las peticiones al servidor.

La forma definir una página que utilice este API es mediante la declaración de un archivo de *manifest* que debe declararse como un atributo del elemento HTML:

```
<html manifest="Pagina.appcache">
```

Aunque también es válido que la extensión sea cualquier otra como *.manifest*. Un archivo de manifiesto puede tener cualquier extensión, pero tiene que ser servido con el tipo MIME correcto: ***text/cache-manifest***. (Es posible que tenga que añadir este tipo mime al servidor Web, si no lo tiene disponible).

```
<html manifest="Pagina.manifest">
```

Este atributo debe estar presente en cada página sobre la que queramos utilizar recursos de caché. Esto significa que cualquier página a la que el usuario navega y que incluye un manifiesto se añade de forma implícita a la caché de la aplicación. Así, no hay necesidad de enumerar todas las páginas de su manifiesto.

El atributo *manifest* puede apuntar a una dirección URL absoluta o relativa, pero, si es una URL absoluta, debe estar bajo el mismo dominio que la aplicación web.

```
<html manifest = "http://www.ejemplo.com/ejemplo.mf">
```

Estructura de un archivo *.manifest*

Un archivo .manifest sencillo, solo requiere de una declaración inicial (CACHE MANIFEST) escrita en mayúsculas, seguida por la lista de ficheros que deseamos almacenar para esa página. Según esto, podría tener la siguiente estructura:

```
CACHE MANIFEST
index.html
stylesheet.css
images/logo.png
scripts/main.js
```

Algunos navegadores imponen restricciones sobre la cantidad de la cuota de almacenamiento disponible para su aplicación. En Chrome, por ejemplo, *AppCache* utiliza un pool compartido de almacenamiento temporal que otras API offline puede compartir. Si estamos escribiendo una aplicación para la Chrome Web Store, utilizando el ***unlimitedStorage*** elimina esa restricción.

Si el archivo de manifiesto o un recurso especificado en ella no se puede descargar, todo el proceso de actualización de la caché falla. El navegador seguirá usando la vieja caché de la aplicación en caso de fallo.

Opciones avanzadas

En ocasiones, lo ideal es poder indicar excepciones al comportamiento predeterminado, *fallbacks* y otros aspectos relacionados. Para estos casos, existen otras posible división de secciones llamadas CACHE, NETWORK y FALLBACK (deben ir en mayúsculas).

CACHE:

Esta es la opción por defecto para las entradas. Los archivos que figuran en esta cabecera (o inmediatamente después del CACHE MANIFEST) se ubicarán explícitamente en caché después de descargarse por primera vez.

NETWORK:

Los archivos que figuran en esta sección son los recursos que requieren una conexión con el servidor. Todas las peticiones de estos recursos deben eludir la caché, incluso si el usuario no está en línea. Se pueden utilizar caracteres comodines.

FALLBACK:

Es una sección opcional que especifica las páginas alternativas si un recurso es inaccesible. La primera URI es el recurso, la segunda, el *fallback*. Ambos URI debe ser relativas y desde el mismo origen que el archivo de manifiesto. Se pueden utilizar caracteres comodines.

Los comentarios se incluyen mediante el símbolo #. Lo que sigue es un modelo que incluye las 3 secciones, indicando los elementos que deben ir a la caché, los que requieren conexión y las alternativas.

```
CACHE MANIFEST
# 2013-07-11:v2

# Recursos principales para la caché
CACHE:
/favicon.ico
index.html
stylesheet.css
images/logo.png
scripts/main.js

# Recursos que requieren estar conectado
NETWORK:
login.aspx
/LogicaNegocio
http://api.twitter.com

# static.html debe servirse si static.aspx no está
disponible
# offline.jpg debe servirse en lugar de cualquier imagen
de /imagenes/grandes/
```

```
# offline.html debe servirse en lugar de  cualquier otro
archivo .html
FALLBACK:
/static.aspx /static.html
images/large/ images/offline.jpg
*.html /offline.html
```

Tenga presente que la caché de una aplicación sólo se actualiza cuando cambia su archivo de manifiesto. Así, por ejemplo, si edita un recurso de imagen o cambia una función de JavaScript, los cambios no actualizarán en el caché. Se debe modificar el archivo de manifiesto e informar al explorador para que actualice los archivos almacenados en caché.

Basta con una línea de comentario con un número generado versión, el hash de los archivos, o una marca de tiempo para garantizar que los usuarios tengan la última versión del software. También se puede actualizar la caché mediante programación, una vez que una nueva versión esté preparada, como indicamos más adelante.

Actualización de la caché

Una vez que la aplicación está *off-line* sigue estando almacenada en caché hasta que se produzca uno de los casos siguientes:

1. El usuario borra el almacenamiento de datos de su navegador para ese sitio.

2. El archivo de manifiesto se modifica. Nota: la actualización de un archivo que aparece en el manifiesto no significa que el navegador recargue ese recurso en la caché. Se debe modificar el archivo de manifiesto en sí.

3. El caché de la aplicación se actualiza por programa.

Estado de caché

El objeto ***window.applicationCache*** habilita el acceso mediante programación a la caché de la aplicación del navegador. Su propiedad

status es útil para verificar el estado actual de la caché, cosa que podríamos comprobar con un fragmento JavaScript como el siguiente:

```
function comprobarCache() {

    var appCache = window.applicationCache;
    switch (appCache.status) {
        case appCache.UNCACHED: // UNCACHED == 0
            return 'noCacheado';
            break;
        case appCache.IDLE: // IDLE == 1
            return 'inactivo';
            break;
        case appCache.CHECKING: // CHECKING == 2
            return 'comprobando';
            break;
        case appCache.DOWNLOADING: // DOWNLOADING == 3
            return 'descargando';
            break;
        case appCache.UPDATEREADY:  // UPDATEREADY == 4
            return 'actualizacionDisponible';
            break;
        case appCache.OBSOLETE: // OBSOLETE == 5
            return 'obsoleto';
            break;
        default:
            return 'desconocido'; // Desconocido
            break;
    };
}
```

Actualización de la caché

Para actualizar la caché mediante programación, primero haremos una llamada a **applicationCache.update()**. Esto intentará actualizar la memoria caché del usuario (lo que requiere que el archivo de manifiesto haya cambiado). Finalmente, cuando la propiedad **applicationCache .status** esté en estado **UPDATEREADY**, otra llamada a **applicationCache .swapCache()** intercambiará la vieja memoria caché por la nueva. El

código algo como esto:

```
var appCache = window.applicationCache;

appCache.update(); // Intento de actualizar la caché.

if (appCache.status ==
window.applicationCache.UPDATEREADY) {
// La lectura ha funcionado e intercambiamos la caché
      appCache.swapCache();
}
```

Ahora bien, esta forma de trabajo no actualiza los recursos en los clientes. Solo le indica al navegador que debe comprobar la existencia de un nuevo manifiesto, descargar el contenido actualizado que especifica, y repoblar la caché de la aplicación. Por lo tanto, se necesitan dos peticiones al servidor para refrescar el nuevo contenido de los usuarios: una para desplegar un nuevo caché de la aplicación, y otra para actualizar el contenido de la página.

Por suerte, podemos evitar este proceso. Para ello, basta con implementar una función que monitorice el evento ***updateready*** del objeto ***appCache*** en la carga de la página:

```
// Comprobar si hay una nueva caché para descarga
window.addEventListener('load', function (e) {


window.applicationCache.addEventListener('updateready',
function (e) {
        if (window.applicationCache.status ==
window.applicationCache.UPDATEREADY) {
  // El navegador detecta una nueva caché.
  // Realiza el intercambio de caché para obtener los
nuevos datos.
            window.applicationCache.swapCache();
            if (confirm('Existe una nueva versión de esta
página. ¿Desesa actualizar?')) {
                window.location.reload();
```

```
            }
        } else {
            // el Manifest no ha cambiado y no hay nada
que recargar.
        }
    }, false);

}, false);
```

Otros eventos del objeto *appCache*

Naturalmente, existe otro conjunto adicional de eventos que permite comprobar el estado de la caché. Si deseamos controlar más exhaustivamente lo que sucede con la caché en cada posible estado, podríamos programar un par de rutinas similares al del siguiente código fuente:

```
function manejadorEventosCache(e) {
    //...Lo que corresponda a cada situación
}

function gestionErrorCache(e) {
    alert('Error: Fallo al actualizar la cache');
};

// Se lanza tras el primer almacenamiento en caché
appCache.addEventListener('cached',
manejadorEventosCache, false);

// Comprobar actualización. Siempre es el primer evento
de la secuencia
appCache.addEventListener('checking',
manejadorEventosCache, false);

// Se ha encontrado una actualización. el navegador está
descargando los recursos.
appCache.addEventListener('downloading',
manejadorEventosCache, false);
```

```
// El manifest devuelve un código 404 o 410: o ha fallado
la descarga,
// o el manifest ha cambiado mientras se descargaba.
appCache.addEventListener('error', gestionErrorCache,
false);

// Se lanza después de la primera descarga del manifest.
appCache.addEventListener('noupdate',
manejadorEventosCache, false);

// Lanzado si el manifest devuelve un código 404 o 410.
// El cache de aplicación se borrará
appCache.addEventListener('obsolete',
manejadorEventosCache, false);

// Se lanza tras la descarga de cada elemento listado en
el manifest.
appCache.addEventListener('progress',
manejadorEventosCache, false);

// Se lanza cuando los recursos del manifest han sido
actualizados.
appCache.addEventListener('updateready',
manejadorEventosCache, false);
```

Finalmente, hay que recordar que si el archivo *manifest* o un recurso especificado en ella no puede descargar, toda la actualización falla. En ese caso, el navegador seguirá utilizando la antigua caché de la aplicación.

IndexedDB: emulación de una base de datos en el cliente

Como apuntábamos más arriba, el problema del almacenamiento local ha sido uno de los caballos de batalla de este nuevo conjunto de API disponibles en HTML 5, y las que acabamos de ver ofrecen soluciones a problemas habituales de las aplicaciones Web con muy

buen rendimiento.

No obstante, las capacidades que hemos mostrado en los sistemas de almacenamiento de ***localStorage*** y ***sessionStorage***, se quedan algo pequeñas cuando deseamos almacenar mayores cantidades de información o necesitamos hacerlo de una manera más estructurada.

Un primer intento: Web SQL

El primer intento de crear una especie de Base de Datos de almacenamiento local fue la propuesta Web SQL, pero, por razones varias, esta propuesta se abandonó. En principio, todos los agentes de usuario que realizaron implementaciones iniciales, se basaron en SQL Lite, pero se requería que la especificación utilizase múltiples implementaciones independientes para poder continuar con el proceso de estandarización.

Ante esa situación, la W3C optó por no continuar con el proceso de confección del estándar y así aparece en el documento oficial "*Web SQL Database*"[92] que lo define como una API para almacenar datos en bases de datos que puedan ser consultadas utilizando una variante del lenguaje SQL. A continuación, se indica que la especificación ha sido abandonada a favor de otras dos propuestas: Web Storage (que incluye las dos propuestas analizadas antes: **localStorage** e **sessionStorage**), e **Indexed DB**.

Estado de la especificación

Se trata de una API para crear bases de datos locales, que añade una estructura más propia de las bases de datos tradicionales. La especificación completa, se encuentra el documento "*Indexed Database API*" en la dirección http://www.w3.org/TR/IndexedDB/ y está ya concluido pues su nivel de madurez es el de "*Candidate Recommendation*", con fecha Julio 2013.

El interés en esta especificación queda patente al comprobar en el equipo

[92] http://www.w3.org/TR/webdatabase/

de desarrollo a miembros de Mozilla, Microsoft, y Google, por lo que se espera que avance con rapidez hacia un adecuado nivel de soporte y se incorpore pronto al conjunto de API utilizables, en todos los navegadores. Al momento de escribir estas líneas (julio/2013), el soporte es bastante completo en IE10/IE11, Chrome y FireFox, no estando soportado por Opera.

Arquitectura

El sistema permite el almacenamiento de valores simples y de objetos jerárquicos, de forma que cada registro está compuesto de una clave y un valor asociado. A este respecto, es similar a los dos anteriores, si bien la idea de *IndexedDB* se centra más en el uso del almacén como base de datos y no como almacén temporal o personal de pequeños conjuntos de datos.

Para ello, el estándar exige la implementación de índices para acceder y manipular los registros guardados (o para cualquier operación con ellos), siendo posible acceder a un registro mediante la clave asociada con él o por su índice. Por la misma razón, podemos controlar aspectos típicos de las bases de datos, como la duplicación de claves.

Resulta especialmente útil para el uso de aplicaciones "*Off-line*", que pueden necesitar mantener grandes cantidades de información, y requerir un acceso rápido a las operaciones habituales.

Mecanismo operativo

El sistema define un conjunto de objetos por cada usuario, que se organizan en forma jerárquica, según vemos en la figura siguiente. Cada usuario dispone de un origen de información (***Origin***), y cada origen puede tener asociadas una o más bases de datos. A su vez, cada Base de Datos está compuesta por uno o más ***almacenes*** u ***object stores***. Y cada almacén, guarda un conjunto de registros en el formato de parejas clave/valor (***key/value pair***).

El gráfico de la figura 11, muestra un esquema de esta arquitectura con sus principales elementos:

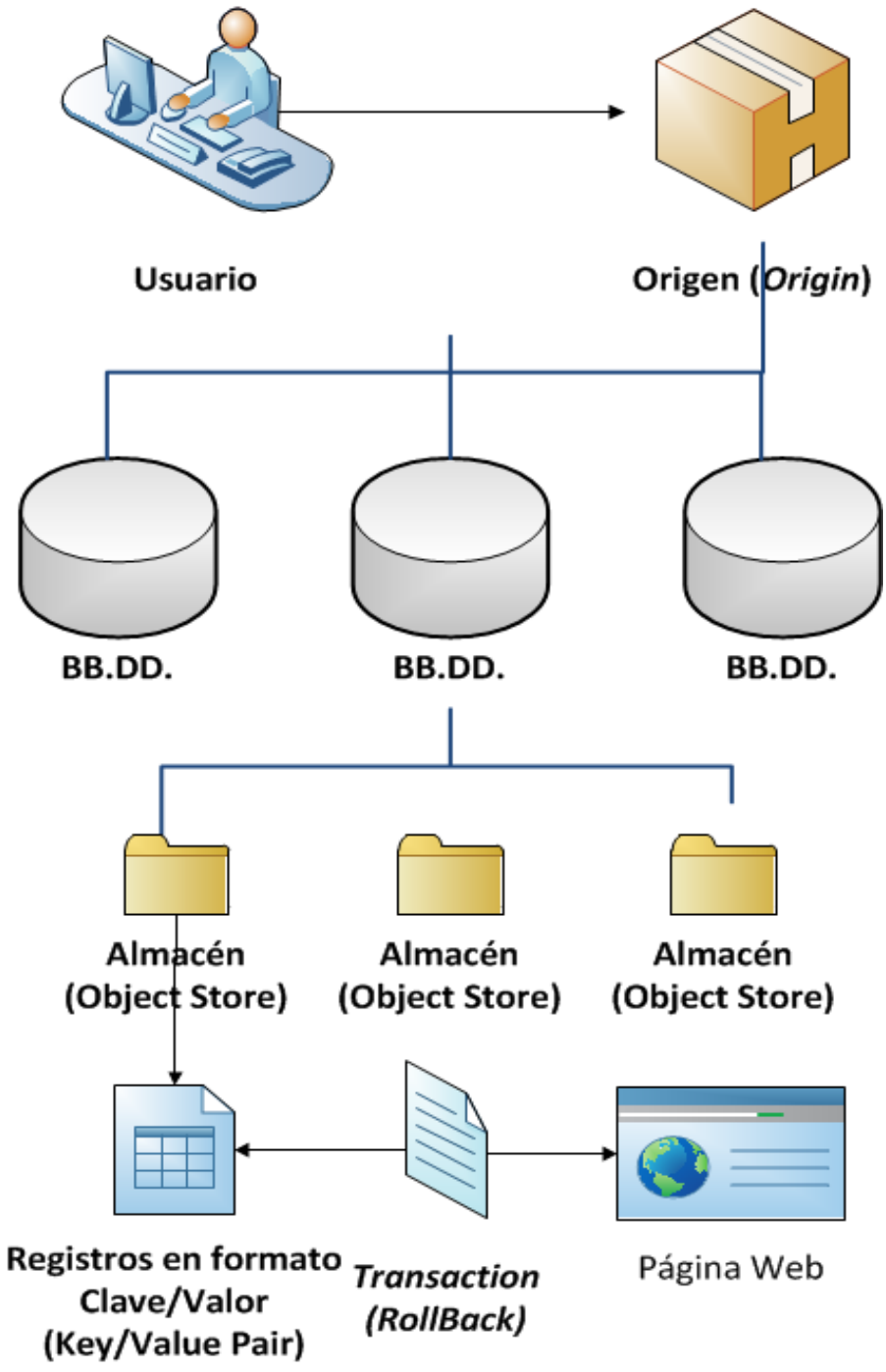


Fig. 11: Esquema organizativo de la arquitectura de IndexDB

Cada interacción con un almacén tiene lugar a través de una transacción (y por lo tanto dispone de capacidades de ***rollback***), y, siguiendo la normativa actual para este tipo de operaciones, se produce mediante un proceso asíncrono.

Esta API es, con mucho una de las más complejas de todas las aparecidas en el estándar (su propósito, también lo es), de forma que para una relación de todos los objetos, métodos, propiedades y eventos relacionados con ella, recomendamos una lectura más detallada de la documentación oficial indicada más arriba, así como de la documentación de las implementaciones que los diversos navegadores han hecho de ella.

En el caso de IE10/IE11, el sitio oficial explica con bastante detalle todos los artefactos disponibles para trabajar con IndexedDB[93], y para *FireFox*[94] también disponemos de documentación propia. Incluso IBM dispone de un artículo bastante detallado, basado –eso sí- totalmente en el documento del estándar[95].

Un primer ejemplo

Vamos a ver un par de ejemplos de la puesta en marcha de esta API dentro de Visual Studio 2013. Supongamos que queremos dotar a una página Web de capacidad de almacenamiento de los comentarios del usuario en un archivo local de éste, y que esos comentarios contienen 3 "campos": *nombre*, *fecha* y *comentario*.

En la parte HTML (la que es significativa para el ejemplo), tenemos un par de elementos para mostrar un artículo y una tabla vacía ("*Comentarios_A1*"), con 3 columnas para recoger el valor de los campos:

```
<h2>IndexedDB. <label id="soporte"></label></h2>
<article>
    <h4>Auctor articulum: Publio Flavio</h4>
```

[93] http://msdn.microsoft.com/en-us/library/ie/hh772651(v=vs.85).aspx

[94] https://developer.mozilla.org/en/docs/**IndexedDB**

[95] http://www.ibm.com/developerworks/library/wa-indexeddb/

```
    <h3>Lorem Ipsum</h3>
    Lorem ipsum dolor sit amet, consectetuer adipiscing
elit,
    sed diam nonummy nibh euismod tincidunt ut laoreet
dolore magna
    aliquam erat volutpat.
</article>
<h5>Comentarios</h5>
<table id="Comentarios_A1">
    <colgroup class="colImpar"></colgroup>
    <colgroup></colgroup>
    <colgroup class="colImpar"></colgroup>
    <thead>
        <tr>
            <td>Nombre</td>
            <td>Fecha</td>
            <td>Comentario</td>
        </tr>
    </thead>

    <tbody></tbody>
</table>
```

En el código JavaScript, lo primero que haremos es comprobar el soporte del funcionamiento del API de ***Indexed DB*** en el navegador, cosa que resolvemos fácilmente con la librería *Modernizr*, instalada manualmente, o mediante el complemento *NuGet* de Visual Studio. Si optamos por esta segunda opción, probablemente, descargaremos la última versión disponible, que, en este caso resulta ser *Modernizr* 2.6.2.

Una vez hecha la referencia a la librería, probamos el soporte con una comprobación del valor de la propiedad *Modernizr* del objeto *window*:

```
// Comprobación de soporte IndexedDB
if (!(Modernizr.indexeddb)) {
    $('#soporte').html("Soporte navegador: NO")
} else {
    $('#soporte').html("Soporte navegador: SI")
}
```

La propiedad ***indexeddb*** del objeto window es, en realidad un objeto de la clase ***IDBFactory*** (tal y como la define el estándar), y es la que permite crear y manipular objetos de una base de datos.

Una vez hecho esto, podemos crear una nueva base de datos para comenzar a hacer pruebas. Para ello, el método ***open*** del objeto ***IndexedDB*** abre una base de datos existente, y si no existe, crea una nueva con el nombre que se le indica como argumento.

Podemos programar una "*closure*" de JavaScript que permita definir el proceso de apertura de la base de datos, así como la carga inicial de algunos comentarios de prueba para ver los resultados después en un proceso de lectura. El código inicial sería el siguiente:

```
// Abrir la base de datos
var refBD = window.indexedDB.open("BDComentarios", 1);

// Rellenar la base con algunos comentarios
refBD.onupgradeneeded = function (e) {
    var db = e.target.result;
    var tabla =
db.createObjectStore("Comentarios_Articulo1", {
autoIncrement: true });
    tabla.createIndex("nombre", "nombre", { unique: false
});

    tabla.add({ nombre: "Obdulio", fecha: new Date(2013,
7, 1).toLocaleDateString(), comentario: "¡Qué
profundo...!" });
    tabla.add({ nombre: "Palmacio", fecha: new Date(2013,
7, 1).toLocaleDateString(), comentario: "¿Profundo? Lee a
los auténticos: Piero Grullo, Kross Pedal..."});
    tabla.add({ nombre: "Filapiano", fecha: new
Date(2013, 7, 2).toLocaleDateString(), comentario: "Pues
a mí no me va el latín." });
    tabla.add({ nombre: "Monglorio", fecha: new
Date(2013, 7, 3).toLocaleDateString(), comentario: "A mí
sí; lo practico en la intimidad..." });

}
```

```
// Consultar la BB.DD. e inicializar la IU
refBD.onsuccess = function (e) {
    var db = e.target.result;
    mostrarComentarios(db);
};
```

En este bloque abrimos una base de datos denominada *BDComentarios*. Como no existe, se crea una nueva y se almacena una referencia a ella en la variable ***refBD***, lo que –en realidad- supone un proceso asíncrono, a cuya finalización se lanzará el evento ***onupgradeneeded*** que se produce cuando se abre una base de datos con un nuevo número de versión (como en este caso).

Aquí, la propiedad ***result*** del objeto ***target*** almacena la referencia a la base de datos, que nos sirve para realizar cualquier operación con ella (y que almacenamos en la variable ***db***).

Con esa nueva referencia podemos crear una tabla (***ObjectStore***), que llamamos "*Comentarios_Articulo1*", y que definimos con un índice en el primer "campo": *nombre*, indicándole que no debe ser única (ya que un usuario puede incluir más de un comentario).

A continuación, añadimos 4 comentarios de prueba al primer artículo (el bloque en latín "*Lorem Ipsum*", que incluimos al principio de la página). Para ello, utilizamos el método ***add*** de este objeto, pasándole los nuevos registros.

Una vez hecho esto programamos la conclusión del proceso asignando al evento ***onsuccess*** (que tiene lugar cada vez que se produce una operación con los contenidos de la BB.DD. se realiza con éxito), para que recoja la nueva referencia a la base de datos y llame a un método que actualice la interfaz de usuario (`mostrarComentarios()`).

Ya que la función recibe como referencia la propia base de datos, podemos programar una transacción sobre ella (objeto ***transaction***). Hay que notar que se requiere la transacción para cualquier operación, ya que es el objeto que permite el acceso al ***ObjectStore***.

Funcionamiento del proceso de lectura

Para realizar el recorrido por los contenidos de la tabla de comentarios, en ***IndexedDB***, utilizamos un objeto ***Cursor***, que inicializamos mediante ***openCursor*** sobre la tabla activa. Este objeto devuelve una referencia a la primera fila del ***objectStore*** y a partir de ella podemos realizar muchas de las operaciones comunes en las bases de datos. El código de este método es el siguiente:

```
// LISTADO COMPLETO
function mostrarComentarios(db) {
    // Vaciamos los posibles contenidos previos.
    var tbody = $("#Comentarios_A1 tbody");
    tbody.empty();

    var transaccion =
db.transaction(["Comentarios_Articulo1"]);
    var tabla =
transaccion.objectStore("Comentarios_Articulo1");

    tabla.openCursor().onsuccess = function (e) {
        var fila = e.target.result;

        if (fila) {
            tbody.append("<tr><td>" + fila.value.nombre +
                "</td><td>" + fila.value.fecha +
                "</td><td>" + fila.value.comentario +
                "</td></tr>");
            // El cursor sigue el recorrido hasta que
            // continue() devuelve false.
            fila.continue();
        }
    };
}
```

Este objeto *cursor* dispone de métodos y propiedades que nos dan acceso a los valores almacenados y a las acciones típicas que son de utilidad al manejar conjuntos tabulares de información: avanzar, continuar, borrar y actualizar.

Entre las propiedades interesantes caben destacar *direction* (permite cambiar la dirección del recorrido), *key* (comprueba si un campo es clave), NEXT y NEXT_NO_DUPLICATE (fila siguiente y siguiente no repetida), PREV y PREV_NO_DUPLICATE (fila anterior y anterior no repetida), *primaryKey* (devuelve la clave primaria), *source* (devuelve una referencia al objeto *ObjectStore* al que pertenece la fila) y finalmente, *value* que es el que contiene las referencias a todos los campos definidos en la tabla.

Podemos ver algunas de esas propiedades en la captura de la figura siguiente dentro del editor de Visual Studio.

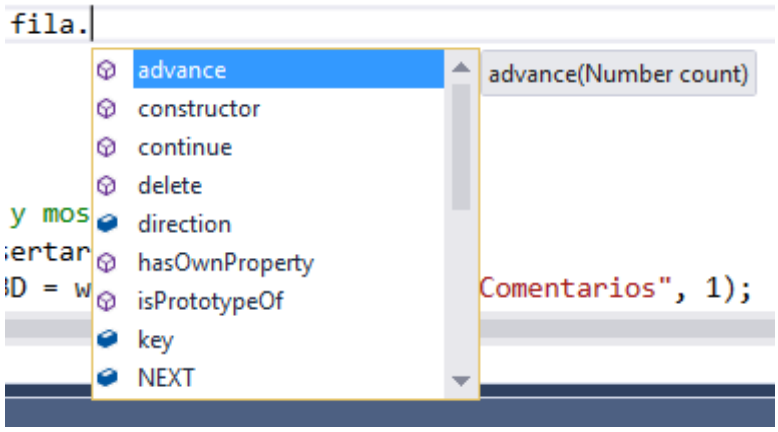


Fig. 12: propiedades de un objeto cursor vistas desde V. Studio 2013

Como siempre, dado que la llamada es asíncrona, encadenamos la programación del evento ***onsuccess***, para que cuando se cree este objeto correctamente, recibamos esa referencia a la primera fila del ***ObjectStore***, y vayamos recuperando los valores de cada fila, y añadiendo su contenido a la tabla vacía inicial, mediante el método ***tbody.append*** de JavaScript.

Filtros

En el caso de que queramos filtrar la información por un criterio concreto, deberemos definir un objeto de rango (*range*) y pasarlo como argumento de la llamada a ***openCursor***, teniendo en cuenta que, previamente,

debemos indicar el índice a usar, ya que estos resultados los obtiene accediendo al conjunto adyacente de registros que cumplen con la clave indicada en el objeto ***range.***

Así pues, si tuviéramos varios comentarios del usuario *Filapiano*, deberíamos de modificar el código de la llamada añadiendo lo siguiente:

```
var indice = tabla.index("nombre");
var rango = IDBKeyRange.only("Filapiano");
indice.openCursor(rango).onsuccess = function (e) {
```

Hay que observar que –en este caso- no es el objeto ***store*** el que abre el cursor, sino el propio índice.

Proceso estándar

La ejecución de ese código en cualquiera de los navegadores modernos (a excepción de Opera), producirá una salida como la siguiente:

Fig.13: Salida del código inicial en IE10/11

(El código restante y de estilos no se incluye por claridad).

Añadiendo información del usuario

Resulta sencillo igualmente añadir un poco de código HTML y escribir una función para que nos permita incluir comentarios dinámicamente. Para ello, en la parte HTML, ampliamos el código del final con lo siguiente:

```
<article id="entradasUsuario">
    <label>Nombre: </label>
    <input type="text" id="txtNombre" value="" required
/><br/>
    <label>Fecha: </label>
    <input type="date" id="txtFecha" value="" required
placeholder="dd/mm/aaaa"/><br />
    <label>Comentario: </label>
    <input type="text" id="txtComentario" value=""
required /><br />
    <input type="button" id="btnNuevoComentario"
value="Añadir" onclick="insertarComentario()" />
</article>
```

Como vemos, 3 elementos ***<input>*** sirven como colectores de la información del usuario. Su contenido es obligatorio y el botón ***Añadir*** tiene programada una llamada al método **insertarComentario.**

Ese método, recogerá la información introducida, volverá a reproducir todo el proceso de apertura de la base de datos, insertará los valores, comprobará que el proceso ha funcionado correctamente y –finalmente- volverá a llamar al método **mostrarComentarios** para actualizar la interfaz de usuario de la página y mostrar los cambios.

```
// Insertar y mostrar cambios.
function insertarComentario() {
    var refBD = window.indexedDB.open("BDComentarios",
1);

    refBD.onsuccess = function (e) {
```

```
        var db = e.target.result;
        var transaccion =
db.transaction(["Comentarios_Articulo1"], "readwrite");
        var tabla =
transaccion.objectStore("Comentarios_Articulo1");
        var nombre = $("#txtNombre").val();
        var fecha = new Date($("#txtFecha").val())
        var comentario = $("#txtComentario").val();

        tabla.add({
            nombre: nombre,
            fecha: new Date(fecha).toLocaleDateString(),
            comentario: comentario
        }).onsuccess = function (e) {
            mostrarComentarios(db);
        };
    }
    refBD.onerror = function (e) {
        alert("Error al insertar");
    }
}
```

Como vemos, tras abrir la base de datos y acceder al *objectStore* recuperamos los valores de los 3 elementos ***<input>*** y los utilizamos como argumentos del método ***add*** de nuestra tabla. Eso es todo.

Para actualiza la IU, programamos (igual que antes), el método ***onsuccess*** para que, cuando se produzca, vuelva a llamar a nuestro método de lectura. Además, y como medida preventiva, hemos programado el evento ***onerror*** de la base de datos, para que si hay algún problema nos indique esa situación.

El lector puede probar este código y recuerde que –como el proceso de creación y borrado de la base de datos es muy rápido- puede incluir al principio de las llamadas JavaScript un pequeño fragmento para borrar la base de datos y volverla a crear cada vez que se carga la página, lo que sería simplemente una llamada de este tipo:

```
// Primero borramos cualquier instancia anterior
// (Solo a efectos de la demo)
var petición =
window.indexedDB.deleteDatabase("BDComentarios");
petición.onsuccess = function (event) {
    return;
}
```

El objeto ***IndexedDB*** posee, además de los métodos ***open*** y ***deleteDatabase*** un método ***cmp*** que permite comparar dos objetos.

Por razones de espacio no tenemos sitio para más, pero con estos ejemplos iniciales puede darse una idea el lector de las posibilidades que ofrece esta API en contextos de negocio y otros donde el mero almacenamiento local básico tipo clave/valor no sirve para nuestras necesidades.

Esta API se completa con la siguiente, que nos permite leer información de archivos locales de la máquina del usuario.

File API: API para acceso a ficheros locales

Con la presencia cada vez mayor de aplicaciones Web (y también de sitios Web avanzados que requieren de opciones más complejas), se echaba en falta un mecanismo que –preservando la seguridad- permitiese a los usuarios manipular sus archivos locales desde una página.

Con esa idea nació File API, que se encuentra en el estado "*Working Draft*"[96] que permite programar acciones sobre uno o varios archivos en la máquina del usuario, no solo para transferencias, sino para procesarlos de alguna forma, o como parte de la gestión de las aplicaciones Web.

[96] http://www.w3.org/TR/FileAPI/

Opciones

En el futuro, posiblemente, esta API nos permita crear, leer, modificar, añadir y borrar archivos como si estuviéramos en una aplicación de escritorio con todos los permisos, *siempre y cuando se utilice con la opción vinculada al* ***Almacenamiento Aislado***. Pero, de momento esto son solo propuestas individuales (como la de Chrome, que no forma parte del estándar).

Lo que sí podemos hacer es trabajar conjuntamente con el elemento ***<input type="file">*** que permite realizar la selección de los archivos a tratar. Además, puede configurarse mediante el atributo ***multiple***, para permitir la selección de más de un archivo cuando se le presente al usuario la Caja de Dialogo de "*Abrir Fichero*".

El conjunto de ficheros seleccionado se almacena en la propiedad ***files*** del elemento ***<input>*** y puede ser manejado posteriormente con código JavaScript. Esta API se completa con el evento ***onchange*** que se produce cuando el usuario cierra la caja de diálogo. El proceso es simple de programar y podemos utilizarlo en muchos contextos distintos.

Básicamente, permite realizar las siguientes acciones:

- Leer datos de archivos seleccionados por el usuario
- Una vez hecha la selección se dispone de varios mecanismos para acceder a su contenido
- Permite utilizar algunos objetos pensados para representar y manipular los datos leídos: ***Blob***, ***File*** y ***FileReader***.
- Estos objetos ponen a disposición del programador un conjunto de métodos nuevos para manipular los datos, siendo los más destacados:
    - ***readAsArrayBuffer***
    - ***readAsBinaryString***
    - ***readAsText***
    - ***readAsDataURL***

Ejemplo inicial

Supongamos que queremos programar una sencilla interfaz que habilite la selección de uno o más archivos por parte del usuario y muestre en la página una lista de algunas de sus propiedades. Podríamos programar lo siguiente:

En la parte HTML5:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
    <title>API de ficheros</title>
    <script src="../scripts/jquery-2.0.3.js"></script>
    <style>
        #selectorArchivos { width: 500px; }
    </style>
</head>
<body>
    <h3>Seleccionar fichero(s)</h3>
    <p><input id="selectorArchivos" type="file" multiple
/></p>
    <h3>Ficheros subidos</h3>
    <ul id="archivosSeleccionados"><li>(no hay ficheros
todavía)</li></ul>
</body>
</html>
```

Y en la parte de JavaScript, incluiremos una función que recorra la colección ***files*** del objeto ***input***, y escriba en pantalla 3 de sus propiedades: el nombre del fichero, su tamaño y su tipo:

```
<script>
    (function () {
        // Recuperamos las referencias a los dos
elementos implicados
        var selector =
document.getElementById("selectorArchivos"),
```

```
            listado = document.getElementById(
"archivosSeleccionados");

        // Lo siguiente recorre los archivos y actualiza
la IU
        function recorrerArchivos(listaArchivos) {
            var item, archivo, archivoInfo;
            listado.innerHTML = "";
            // En la zona de declaración de variables del
bucle for,
// declaramos dos variables: la de recorrido (i) y la de
// límite (numArch = numero archivos)
            for (var i = 0, numArch =
listaArchivos.length; i < numArch; i++) {
                item = document.createElement("li");
                archivo = listaArchivos[i];
                archivoInfo =
"<div><strong>Fichero:</strong> " + archivo.name +
"</div>";
                archivoInfo +=
"<div><strong>Tamaño:</strong> " + archivo.size + "
bytes</div>";
                archivoInfo +=
"<div><strong>Tipo:</strong> " + archivo.type + "</div>";
                item.innerHTML = archivoInfo;
                listado.appendChild(item);
            };
        };

        /* Cuando el usuario haya seleccionado algún
archivo,
           se produce el evento -onchange- */
        selector.onchange = function () {
            recorrerArchivos(this.files);
        };

    })();
</script>
```

El soporte está bastante avanzado, y podemos probarlo correctamente

en los navegadores IE10, IE11, FireFox, Chrome y Opera. La salida de este ejemplo en IE11 después de seleccionar un par de archivos gráficos es la que aparece en la figura 14:

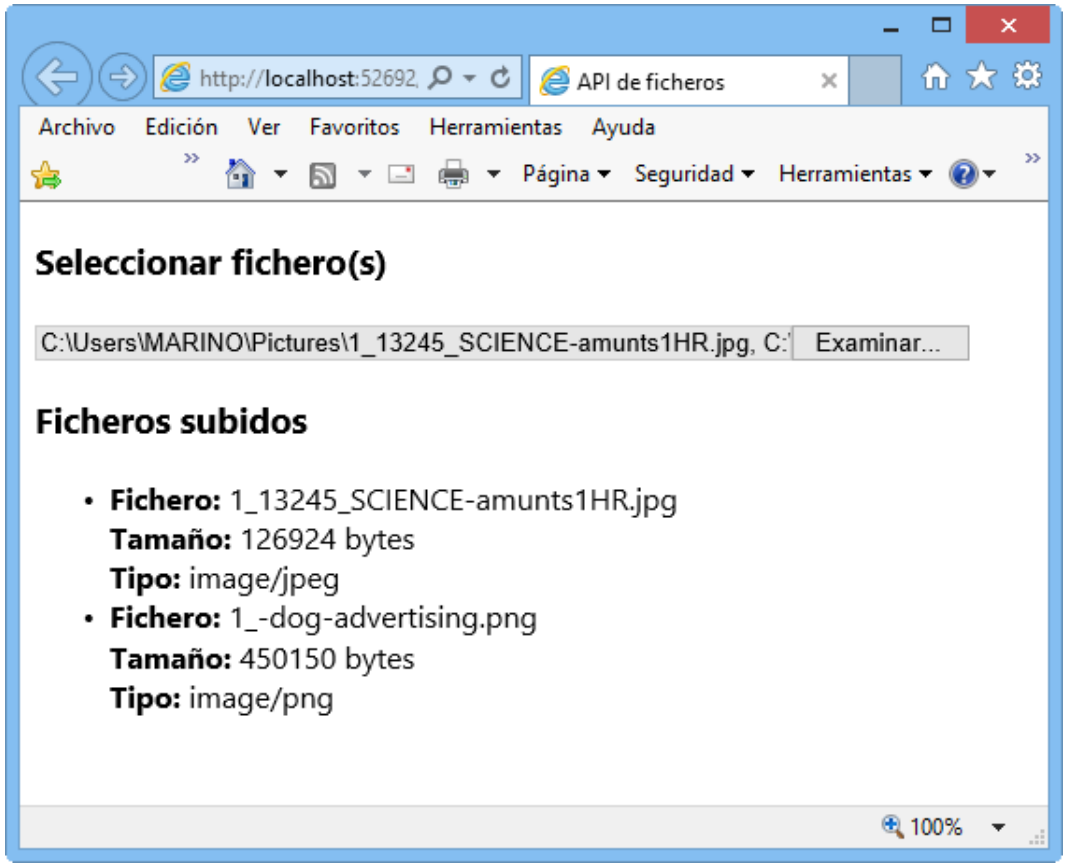


Fig. 14: Salida del código anterior en Internet Explorer 11

Ejemplo 2: Leyendo el contenido de un archivo

Supongamos ahora que queremos leer el contenido de los archivos cargados siempre que estos sean de texto o gráficos, y mostrar su contenido dentro de la página. La estrategia sería que, una vez obtenidas las referencias a los archivos, determinásemos el tipo *mime* para saber el método de lectura a utilizar (y la forma de presentarlo en pantalla), y utilizásemos algún elemento HTML adecuado para la tarea.

Si se trata de un archivo de texto, nos valdrá cualquier elemento capaz de albergarlo. Para el caso de los gráficos, podemos utilizar un elemento ***<canvas>***.

Como el mecanismo de selección ya está disponible, solo tenemos que modificarlo ligeramente para admita únicamente los tipos *mime* que vamos a permitir: "*text/plain*" para los archivos de texto, y "*image/jpeg*" e "*image/png*" para los gráficos. Esto lo conseguimos modificando ligeramente la definición de nuestro elemento ***<input>***, al que –ahora- le podemos indicar una lista enumerada de tipos mime:

```
<input id="selectorArchivos" type="file" multiple
accept="text/plain, image/jpeg, image/png" />
```

Si solo de tratase de imágenes y admitiésemos cualquier tipo de extensión del archivo gráfico, podríamos expresar lo mismo de forma más abreviada:

```
<input id="selectorArchivos" type="file" multiple
accept="image/*" />
```

Esta nueva opción, nos sirve –además- para añadir un elemento de validación en las entradas del usuario.

De acuerdo con esto, tendríamos que convertir nuestra función de lectura añadiendo un mecanismo de selección que discrimine las entradas correspondientes a ficheros de texto de aquellas que son gráficas, y escribir dos funciones separadas para cargar el contenido de unos y otros.

Téngase en cuenta que todas estas nuevas API descansan en mecanismos no bloqueantes y, por lo tanto, suponen la puesta en marcha de procesos asíncronos.

De acuerdo con esto, nuestro método **recorrerArchivos** adoptaría ahora la siguiente forma:

```
// Lo siguiente recorre los archivos y actualiza la IU
function recorrerArchivos(listaArchivos) {
```

```
    var item, archivo, archivoInfo;
    listado.innerHTML = "";
    // En la zona de declaración de variables del bucle
for
    // declaramos dos variables: la de recorrido (i) y la
de límite (numArch)
    for (var i = 0, numArch = listaArchivos.length; i <
numArch; i++) {
        item = document.createElement("li");
        archivo = listaArchivos[i];
        archivoInfo = "<div><strong>Fichero:</strong> " +
archivo.name + "</div>";
        archivoInfo += "<div><strong>Tamaño:</strong> " +
archivo.size + " bytes</div>";
        archivoInfo += "<div><strong>Tipo:</strong> " +
archivo.type + "</div>";
        item.innerHTML = archivoInfo;
        listado.appendChild(item);

        // Aquí, según el tipo mime, leemos el
        // archivo mediante una función u otra.
        if (archivo.type === "text/plain") {
            cargarTextoArchivo(archivo);
        } else if (archivo.type === "image/jpeg" ||
archivo.type === "image/png") {
            cargarArchivoGrafico(archivo);
        }
    };
};
```

O sea, en función del tipo de archivo llamaremos a ***cargarTextoArchivo*** o a ***cargarArchivoGráfico***. Primero, escribimos los 2 métodos necesarios para la lectura de archivos de texto (como toda esta API es asíncrona[97], un método configura la llamada y el otro la recepción y actualización de

[97] En realidad, existe una versión de esta API que es síncrona, pero solo puede ser usada en el contexto de un Web Worker, no dispone de funciones *callback* y algunos métodos de llamada tienen un nombre distinto para hacerse eco de esta situación. La especificación oficial indicada antes explica estas diferencias.

la IU. En el caso de archivos de texto, los dos métodos adoptarán la siguiente forma:

```
function cargarTextoArchivo(archivo) {
    var reader = new FileReader();
    // Manejadores asíncronos
    // "onloadend" se lanza cuando los contenidos
    // han sido cargados correctamente en memoria.
    reader.onloadend = mostrarContenidoTexto;
    if (archivo) { // Asegurar que hay archivo.
        // Comienza la lectura asíncrona
        reader.readAsText(archivo);
    }
}

function mostrarContenidoTexto(evt) {
    var contenido = evt.target.result;
    // Añadimos el contenido al listado
    $("<p>" + contenido +
"</p>").appendTo("#archivosSeleccionados");
}
```

La función **readAsText** lee el contenido del objeto *File* que recibe como argumento, y –cuando termina- el objeto *reader* ejecuta la función asignada al evento ***onloadend*** si es que existe alguno. La información leída se transporta en la propiedad ***result*** que pertenece al objeto evento que se recibe como argumento. Además, ***readAsText*** admite argumentos adicionales para permitir especificar en tipo de codificación que se desea utilizar en la lectura del fichero.

Finalmente, solo tenemos que añadir ese contenido a la lista inicial, para lo que usamos el método ***appendTo*** de jQuery.

Para el caso de archivos gráficos, el proceso es muy similar, solo que cambiaremos el método de lectura, y usaremos ***readAsDataURL***, para leer el contenido (hemos visto el formato de ***dataURL*** en el apartado de CSS3, a propósito de la posibilidad asignar directamente el contenido de un gráfico a una propiedad mediante esta técnica). El segundo método, utilizará una técnica similar al anterior para actualizar la interfaz de

usuario.

El código correspondiente a estos otros 2 métodos sería el siguiente:

```
function cargarArchivoGrafico(archivo) {
    var reader = new FileReader();
    // Manejadores asíncronos
    // "onloadend" se lanza cuando los contenidos
    // han sido cargados correctamente en memoria.
    reader.onloadend = mostrarContenidoGrafico;
    if (archivo) { // Asegurar que hay archivo.
        // Comienza la lectura asíncrona
        reader.readAsDataURL(archivo);
    }
}

function mostrarContenidoGrafico(evt) {
    $("<img src='" + evt.target.result + "' />")
        .appendTo("#archivosSeleccionados");
}
```

Si el lector tiene curiosidad por ver exactamente qué devuelve la propiedad ***result*** en el caso de un gráfico, aprovecho para recordar que los depuradores disponibles (tanto de Visual Studio, como de IE, Chrome o FireFox), nos lo pueden mostrar sin problemas.

En el caso de que no tengamos V. Studio a mano, la herramienta de desarrolladores (F12) de IE (v11, en mi caso), nos permite poner un punto de ruptura en la sentencia que contiene el resultado y desglosar todas las propiedades y sus valores en tiempo de ejecución en la ventana adyacente, como podemos ver en la figura adjunta:

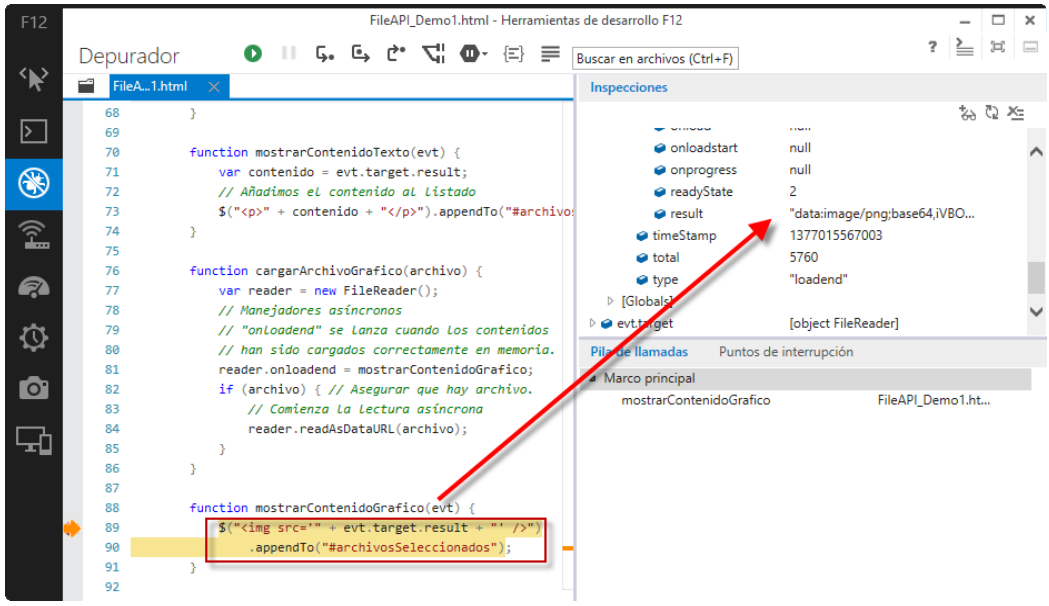


Fig. 15: El depurador de IE11 en funcionamiento mostrando el tipo de dato devuelto por el método readAsDataURL.

La salida final de la ejecución de esta página cuando seleccionamos un archivo de texto y un par de gráficos es la que muestra la figura siguiente:

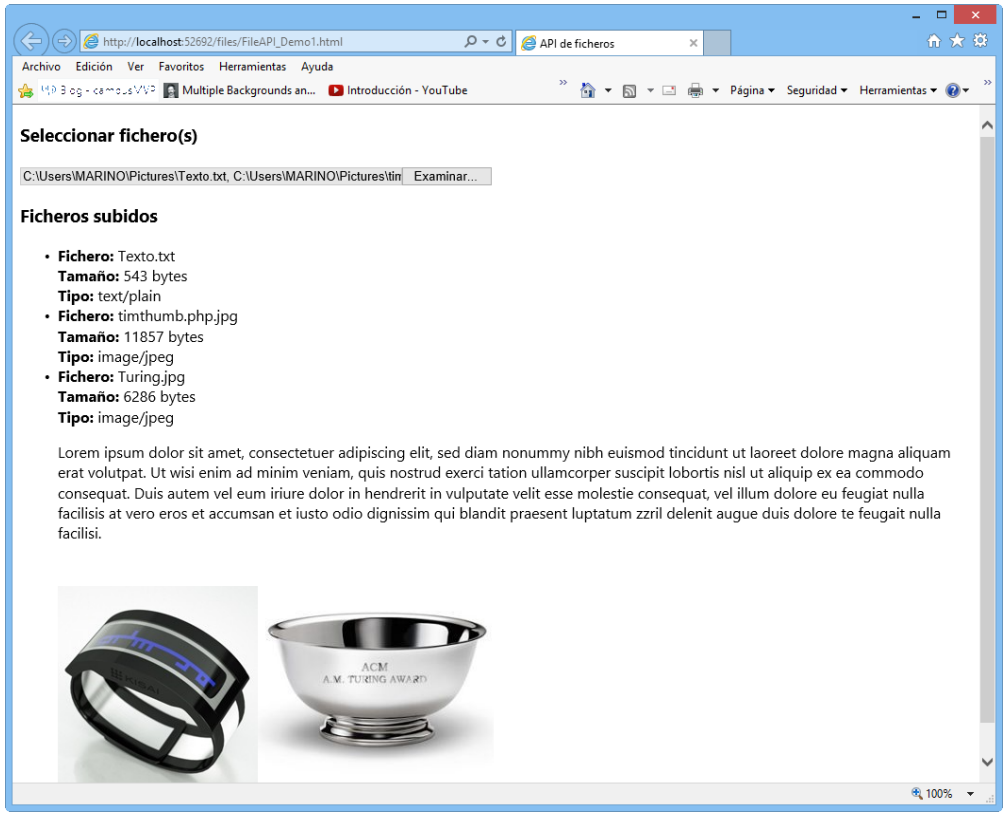


Fig. 16: Salida de la ejecución del código anterior.

Una nota sobre el formato *Data URL*

Se trata de un formato soportado por todos los navegadores que son objeto de esta obra: IE9/IE10/IE11 (de hecho IE8, lo soporta también), Chrome, FireFox y Opera, y está basado en el estándar ***XML Schemas***, que define un tipo de datos ***base64*** para codificar archivos binarios serializándolos como una cadena de bytes, de forma que puedan ser transportados con facilidad por los canales de comunicación. En este caso, se admiten más formatos, siendo su sintaxis la siguiente:

```
data:<tipo-mime> [;base64], <datos>
```

Hay una diferencia: mientras que las URL de archivos del sistema o las Blob URL son referencias al contenido, las Data URL **son** el contenido.

Esto elimina peticiones posteriores al servidor en muchos casos, tal y como veíamos en el apartado de CSS3 con la sintaxis que permite asignar un valor Data URL a una propiedad gráfica como una imagen.

Las API para la mejora del rendimiento de las aplicaciones

Aparte de las consideraciones que hagamos respecto al buen diseño y a una arquitectura adecuada de la información, es preciso que el usuario no asocie ciertas acciones con tiempos de espera, y esto es especialmente notable en el caso de los dispositivos móviles. El logo que acompaña al título de este apartado se refiere genéricamente a "API para la mejora del rendimiento de las aplicaciones", y es una propuesta global de la W3C que se desglosa en varias propuestas individuales: algunas existentes (si bien mejoradas en esta versión, como XHR, o más comúnmente, AJAX), y otras, totalmente nuevas.

Dentro de este apartado, existe un grupo de API que facilitan de forma notable estos principios: el uso de AJAX y JSON para llamadas asíncronas a servicios Web, los procesos asíncronos en la máquina local, mediante el API de *Web Workers*, las comunicaciones dúplex asíncronas y estables entre cliente y servidor que ofrece el API *Web Sockets*, las aplicaciones "off-line" y el uso de cachés locales, los micro-formatos y algunas otras. Comenzamos este apartado revisando los mecanismos de llamadas asíncronas mediante AJAX y JSON.

AJAX: Llamadas asíncronas a un servicio mediante AJAX y JSON

La Wikipedia, define AJAX diciendo que "Ajax, acrónimo de Asynchronous JavaScript And XML (JavaScript asíncrono y XML), es una técnica de desarrollo web para crear aplicaciones interactivas o RIA (Rich Internet Applications). Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador de los usuarios mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma

es posible realizar cambios sobre las páginas sin necesidad de recargarlas, lo que significa aumentar la interactividad, velocidad y usabilidad en las aplicaciones."

AJAX, es por tanto, una tecnología asíncrona, porque los datos que se solicitan al servidor después de la primera carga de la página se descargan en segundo plano, y no resultan bloqueantes para la interfaz de usuario. El soporte de AJAX está garantizado en todos los navegadores actuales desde hace varios años, y descansa en un objeto JavaScript denominado ***XMLHttpRequest***. A pesar de su nombre, las peticiones AJAX pueden recibirse en varios formatos distintos, de los que XML es solo una opción. La otra opción disponible, y cada vez es más utilizada en la red, es el formato de datos JSON (*JavaScript Object Notation*).

El esquema gráfico siguiente (Figura 17) ilustra la arquitectura de las llamadas a servicios mediante este objeto desde JavaScript:

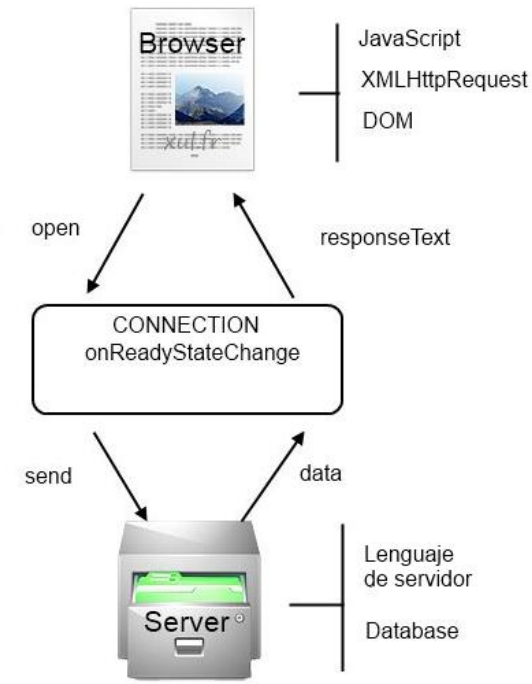


Fig. 17: Esquema de una llamada XMLHttpRquest

La especificación oficial está siendo igualmente administrada por la W3C y existe un documento de enero de 2012 del tipo "*Working Draft*" sobre el nivel 2 de este estándar disponible en la Web[98].

El mecanismo de funcionamiento es sencillo y podemos asumirlo como un proceso en 4 pasos principales:

1) El cliente realiza una petición al servidor mediante una llamada al método ***Send({Lista de Args.})***, que puede contener una lista de argumentos, como podría ser el nombre de un fichero a subir al servidor o una lista de parámetros para autenticación o configuración de la petición.
2) Cuando el servidor atiende la petición, primero examina los argumentos, y la cabecera de la petición. Si no se pasan argumentos (se envía un valor ***null***), el servidor comprueba si tiene sentido para él esa petición, y, si es así, devuelve al cliente la información solicitada (en uno de los formatos válidos comentados).
3) Cuando la información es recibida por el cliente, se dispara un evento que le indica que los datos están listos para ser procesados. Este evento, en realidad, pueden ser dos: ***onreadystatechanged*** y ***onload*** dependiendo de que se trate del soporte del nivel más básico (*Level 1*) o del más actualizado *Level 2*.
4) En último lugar, la información recibida es procesada en el cliente, y el DOM es modificado para reflejar los cambios, y se actualiza la interfaz de usuario, pero solo de los nuevos datos o elementos recibidos y no de la página completa.

Previo a la petición, se habrá configurado adecuadamente el objeto ***XMLHttpRequest***, especificando en su método ***open()***, el modo en que realizamos esta (GET, POST,..) y la URL destinataria de la misma. Cuando se recibe la respuesta del servidor, además, algunas de las propiedades

[98] http://www.w3.org/TR/XMLHttpRequest2/

del objeto ***request*** se modifican, por lo que podemos comprobar sus nuevos valores para garantizar si la petición ha tenido éxito.

El código fuente, por tanto, puede adoptar dos versiones, según se trate de una llamada mediante el sistema más antiguo o la que utilizamos para los navegadores más actualizados. En el primer caso, tendríamos un código JavaScript similar al siguiente:

```
// Esta funcion utiliza XMLHttpRequest Level 1,
// para dar soporte a los navegadores antiguos
function leerVentas_Nivel_1() {
    var url =
"http://localhost:2439/DemosJS5/ventas.json";
      var request = new XMLHttpRequest();
      request.open("GET", url);
      request.onreadystatechange = function() {
if (request.readyState == 4 && request.status == 200) {
                  actualizarIU(request.responseText);
            }
      };
      request.send(null);
}
```

La idea es que la URL nos devuelve un contenido en formato JSON (en este caso apunta a un archivo con esa extensión, pero podría ser cualquier URL válida que respondiera en ese formato).

Como vemos, hemos asignado una función anónima al evento ***onreadystatechange***, para que, al producirse, compruebe lo sucedido con ciertos valores del objeto ***request***, (***readystate == 4***, significa que la respuesta está completa y que podemos acceder a los datos recibidos, y ***request.status == 200***, que todo fue correctamente en el servidor).

Los valores posibles de ***readystate*** son:

Explicación	Valores
Forma en que *readyState* devuelve el objeto	0 = sin inicializar 1 = abierto 2 = cabeceras recibidas 3 = cargando 4 = completado D a un elemento

Por su parte, los valores posibles la propiedad ***status*** son muchos ya que devuelve el código proporcionado por el servidor, siendo 200 el indicador de que todo marchó correctamente.

En la versión más actualizada (*Level 2*), es posible omitir algunas de estas comprobaciones, y reducir el proceso al control del evento ***onload*** que indica que se han cargado los datos de vuelta y están listos para procesarse. El código de este segundo tipo de llamada adoptaría una forma como la siguiente:

```
function leerVentas_Nivel_2() {
    var url =
"http://localhost:2439/DemosJS5/ventas.json";
    var request = new XMLHttpRequest();
    // Opcionalmente, podemos indicar el formato de
    // la respuesta.
    request.responseType = "application/json";
     request.open("GET", url);
     request.onload = function() {
          if (request.status == 200) {
               actualizarIU(request.responseText);
          }
     };
     request.send(null);
}
```

Y, en ambos casos, la propiedad ***responseText*** del objeto ***request*** almacena la información recibida.

Por su parte, la función que actualiza la interfaz de usuario (***actualizarIU***), procedería a analizar los datos recibidos, convertirlos a un objeto manejable por JavaScript, utilizando el método ***parse*** del objeto ***JSON***, y recorrer la información para actualizar el DOM de la página, como vemos en el código siguiente:

```
function actualizarIU(respuestaJSON) {
    var DivVentas =
document.getElementById("DIV_ventas");
    var ventas = JSON.parse(respuestaJSON);
// Tras el "parsing" disponemos de un array con la
//información devuelta
     for (var i = 0; i < ventas.length; i++) {
          var venta = ventas[i];
          var div = document.createElement("div");
          div.setAttribute("class", "FormatoVenta");
div.innerHTML = "Num. Cliente: " + venta.Id + " -
Ciudad: " + venta.Ciudad + " - Total ventas: " +
venta.TVentas;
          DivVentas.appendChild(div);
     }
}
```

La clave del proceso de conversión está el método ***parse*** del objeto ***JSON***, que forma parte del núcleo de JavaScript y se encarga del proceso de conversión de los datos recibidos en un objeto JavaScript manejable por este lenguaje directamente.

Una nota sobre JSON como formato ligero de datos

JSON fue ideado por **Douglas Crockford**[99] cuando trabajaba como arquitecto de la fundación Mozilla, para hacer más fácil la evaluación de un bloque de datos de respuesta desde un servidor utilizando un formato

[99] Ver la especificación oficial en la página: http://www.json.org/json-es.html

similar al que usa el propio lenguaje en la declaración de objetos. De hecho, como ya hemos indicado, es posible utilizar la función ***eval()*** de JavaScript para convertir un bloque de datos JSON en una estructura directamente utilizable por este lenguaje, si bien es una práctica que no se recomienda por razones de seguridad.

No es una notación pensada para eliminar la verbosidad inherente al formato XML (vienen a ocupar lo mismo, y, una vez comprimidos, incluso XML puede resultar más liviano), sino para facilitar la conversión del formato y su uso casi inmediato. Eso es lo que ha hecho que la popularidad de JSON creciera como la espuma. Según la propia Wikipedia, un formato de datos JSON simple puede representarse con el siguiente código fuente:

```
{"menu": {
   "id": "file",
   "value": "File",
   "popup": {
     "menuitem": [
       {"value": "New", "onclick": "CreateNewDoc()"},
       {"value": "Open", "onclick": "OpenDoc()"},
       {"value": "Close", "onclick": "CloseDoc()"}
     ]
   }
 }}
```

Y su equivalente en XML sería algo así:

```
<menu id="file" value="File">
  <popup>
    <menuitem value="New" onclick="CreateNewDoc()" />
    <menuitem value="Open" onclick="OpenDoc()" />
    <menuitem value="Close" onclick="CloseDoc()" />
  </popup>
</menu>
```

Como vemos, la notación permite estructurar en árbol los elementos (igual que XML), ubicándolos en jerarquías delimitadas por símbolos de

llaves (***{}***), que albergan conjuntos de pares de datos en formato clave/valor, donde la clave se expresa entre comillas dobles y se separa mediante el símbolo de dos puntos (**:**) del valor, que puede adoptar diferentes notaciones según su tipo de contenido (la página antes indicada en la nota al pie, explica suficientemente todos los casos).

En el caso de que éste sea un *array* (como el elemento *MenuItem* del ejemplo), estos se representan incluidos entre corchetes (***[]***) que incluirán tantos bloques de llaves conteniendo los pares clave/valor como dimensiones tenga el *array*.

En una notación más habitual, podemos encontrarnos conjuntos de datos que expresen consultas de una tabla en este formato, como podemos ver en el siguiente código de una supuesta tabla de Clientes, donde se indica su identificador, su procedencia y un total de ventas:

```
[{ "Id":100, "Ciudad":"Valladolid","TVentas":65292 },
 { "Id":101, "Ciudad":"Madrid","TVentas":83444 },
 { "Id":102, "Ciudad":"Palencia","TVentas":11345 },
 { "Id":103, "Ciudad":"Sevilla","TVentas":42560 },
 { "Id":104, "Ciudad":"Oviedo","TVentas":18100 },
 { "Id":105, "Ciudad":"Barcelona","TVentas":53040 },
 { "Id":106, "Ciudad":"Pamplona","TVentas":54060 },
 { "Id":107, "Ciudad":"Malaga","TVentas":71550 },
 { "Id":108, "Ciudad":"Salamanca","TVentas":20200 }
]
```

Aunque ya hemos comentado que no se trata de una práctica recomendable, la función ***eval()*** de JavaScript podría evaluar este contenido de forma directa. Esto no es recomendable, y los navegadores utilizan hoy día el objeto ***JSON***, que dispone, de otro método contrario (***stringify***), que convierte objetos JavaScript en una cadena formateada con esta notación.

Según todo lo anterior, hemos aplicado a la salida un archivo de extensión CSS para formatear los elementos creados dinámicamente, de manera que obtendríamos una salida como la que se ve en la figura 18 al abrir el archivo con cualquiera de los navegadores:

Fig. 18: Salida del código anterior en IE9/10/11

En el entorno de desarrollo de Microsoft, hace años que disponemos de una implementación avanzada de esta tecnología, ASP.NET AJAX, que permite trabajar de forma muy sencilla con las llamadas asíncronas gracias al concurso de varios controles de servidor, específicamente, ***asp:scriptmanager*** y ***asp:updatepanel***, que simplifican considerablemente los mecanismos de llamada, recepción y actualización de la interfaz de usuario.

Cabe notar que la primera versión de este componente fue desarrollada por Microsoft para su versión 5.0 del navegador Internet Explorer. Estuvo disponible en forma de componente ActiveX hasta la versión 7, a partir de la cual incluyó el soporte nativo sin necesidad de usar complementos. Todos los navegadores mencionados aquí ofrecen igualmente soporte desde hace varias versiones, por lo que podemos utilizar esta técnica sin necesidad de mecanismos de "*fallback*".

La API *Web Workers*: Tareas asíncronas en el cliente

Una de las funcionalidades que más se demandaban por parte de los programadores Web era la posibilidad de manejar procesos en segundo plano. La API implicada en esta oferta funcional es ***Web Worker***, que permite definir una tarea en JavaScript, e indicar cuál será la función que procesará la información en segundo plano.

El soporte de los objetos *Web Worker* es dispar todavía, aunque la mayoría de los navegadores actuales han incorporado este objeto en su implementación del motor de JavaScript. Para comprobar su existencia debemos hacer una llamada al objeto ***Worker***, que nos devolverá un valor ***undefined*** si no está disponible.

Así pues, la primera llamada debiera ser esa en el código fuente:

```
if (Worker == "undefined") {
   //Actualizar la interfaz de usuario
}
```

Una vez hecho esto, podemos configurar el objeto *Worker*. El primer paso, sería obtener un objeto de este tipo y asignarlo a una variable de trabajo. En el constructor, le pasaremos al objeto el nombre del fichero JavaScript que almacena la tarea a realizar en segundo plano.

El código asociado sería algo así:

```
var worker = new Worker("tarea.js");
```

El objeto ***Worker*** pertenece al motor de JavaScript, y dispone de métodos y propiedades específicas, como el método ***postMessage()***, que permite enviar conjuntos de datos a la tarea en forma de parámetros, y los eventos ***onmessage*** y ***onerror*** que podemos configurar para cuando se produzca la respuesta del evento, o en el caso de un error.

Tanto en la parte de cliente, como en la de servidor, la llamada a ***postmessage(argumentos)*** es la encargada de enviarlos en la dirección opuesta, solo que en el caso de la respuesta, el dato propiamente dicho lo recogemos en la propiedad ***data*** del objeto ***event*** que se recibe como parámetro.

Imaginemos que disponemos de una interfaz de usuario muy simple que consta de un solo elemento *** *** de nombre entro del cuerpo del documento, de acuerdo con el siguiente código:

```
<!DOCTYPE html>

<html>
<head>
    <title>Pruebas con el objeto Worker</title>
    <meta charset="utf-8">
    <script src="gestorWorkers.js"></script>
</head>
<body>
    <p id="MensajeDelWorker"></p>
</body>
</html>
```

Para configurar la llamada podemos utilizar un código JavaScript como el siguiente:

```
window.onload = function () {
    if (Worker == "undefined") {

document.getElementById("MensajeDelWorker").innerHTML =
   "Web Workers no disponibles";
    }
    else {
      var worker = new Worker("tarea.js");
      // Esto establece el parámetro que se pasa al
Worker
      worker.postMessage("Datos");
```

```
      // Cuando el proceso termina se produce el evento
onmessage
      worker.onmessage = function (event) {
          var message = "Mensaje del Worker: " +
event.data;
          // Actualiza la interfaz de usuario

document.getElementById("MensajeDelWorker").innerHTML =
message;
      }
      // Además programamos una potencial situación de
error
      worker.onerror = function (error) {

document.getElementById("MensajeDelWorker").innerHTML =
                  "Hubo un error en el fichero " +
error.filename +                    " Línea nº " +
error.lineno + ": " + error.message;
        };
    }
}
```

Y el código correspondiente a la tarea que se pasa como argumento y que está contenido en el fichero "tarea.js", podría ser algo como esto:

```
// Asignamos la función a ejecutar
onmessage = mensajeVuelta;

function mensajeVuelta(event) {
    // El objeto event transporta la información
    // en su propiedad data
    if (event.data == "Datos") {
        postMessage("Datos de vuelta");
    }
    else {
        // Error intencionado
        1/x;
    }
}
```

Donde hemos programado una sencilla bifurcación de error para poder comprobar el funcionamiento de ambas opciones en caso de que no pasemos como argumento la cadena "Datos".

El soporte es bastante completo en todos los navegadores actuales.

La API Web Sockets

De las API dedicadas a comunicaciones, sin duda, la que mayor popularidad y soporte está ganando en la actualidad es ésta.

Wikipedia define la iniciativa Web Sockets como "una tecnología que proporciona un canal de comunicación bidireccional y *full-dúplex* sobre un único socket TCP". Aunque está pensada para ser implantada en navegadores y servidores Web, la idea es que pueda utilizarse en cualquier aplicación cliente/servidor.

Paul Cotton, "*Working Chair*" de W3C y una de las personas que dirige el proceso de estandarización nos comentaba en una entrevista[100] que esta API se trata, en realidad de dos sentimientos encontrados: por una parte –decía- "*si me preguntan cuál es el API más atractiva de todas las presentadas como parte del estándar, diría que Web Sockets, pero si me preguntan igualmente cuál es la más problemática, (o la más peligrosa), respondería igualmente que Web Sockets*". Esto da una idea de las altas expectativas que existen en torno a esta propuesta, y –al mismo tiempo- del enorme cuidado que se está poniendo en su diseño, sus test y su publicación final[101]. En su comentario, Cotton hacía referencia a la

[100] La entrevista está disponible en vídeo en el sitio Channel9: http://channel9.msdn.com/Blogs/channel9spain/Entrevista-a-Paul-Cotton

[101] En este momento existe una especificación de nivel "Draft" (Borrador) con fecha julio 2013, que puede verse en la dirección http://dev.w3.org/html5/websockets/

situación que se produjo cuando un analista de seguridad descubrió un posible "agujero" que obligo a reescribir una de las versiones.

Y es que, a partir de una modificación en el borrador del protocolo[102], se rompió la compatibilidad con *proxies* y pasarelas, lo que le convirtió en vulnerable a ataques del tipo *HTTP smuggling*, como se informa en varias fuentes y explica con detalle la página "*HTTP Request Smuggling*" de "*The Open Web Application Security Project*" (https://www.owasp.org/index.php/HTTP_Request_Smuggling).

Además, hay dos organizaciones implicadas en la construcción de esta API: la propia W3C y la *Internet Engineering Task Force* (IETF)[103], que se encarga de la normalización del protocolo, que han acordado la definición de dos versiones, (***ws://*** y ***wss://***) según se trate de conexiones estándar o cifradas.

Por fortuna, en la situación actual, ese problema ha sido resuelto y la implementación sugerida goza de todas las garantías en ese sentido (si es que en la red, algo puede gozar de todas las garantías de seguridad).

Por otra parte ya existen varias propuestas en forma de librerías que aprovechan esta API y permiten implementar soluciones basadas en este escenario de comunicación que se presenta muy prometedor.

La arquitectura de comunicación

La iniciativa de WebSockets no es la primera que se sugiere para implementar un mecanismo de tipo "*push*" entre servidores y clientes (el proceso mediante el cual es el servidor el que se comunica con un conjunto de clientes para enviarles alguna información.

En la actualidad se están simulando este tipo de cosas mediante mecanismos diversos que tienen diversos inconvenientes,

[102] En su versión 76, para ser exactos

[103] Existe un documento publicado por esta entidad donde se explica profusamente y con gran detalle la arquitectura y funcionamiento del protocolo en su estado actual (ver http://tools.ietf.org/html/draft-ietf-hybi-thewebsocketprotocol-17)

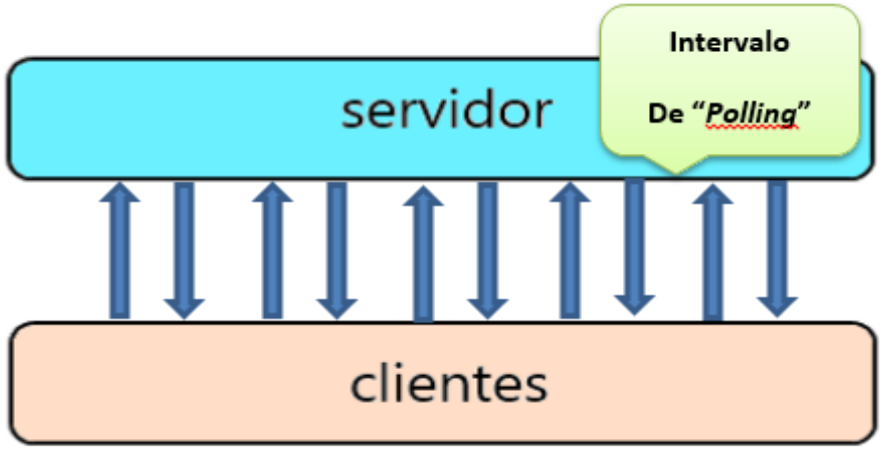
principalmente, debido a que el protocolo de comunicaciones HTTP es un protocolo de petición/respuesta, que no está diseñado para este tipo de comunicaciones.

Las técnicas "*push*" utilizadas en la actualidad

Las técnicas más habituales que se estaban utilizando en la actualidad, incluyen:

- **Solicitud Periódica (*Periodic Polling*)**: Consiste en pedir al servidor que compruebe si tiene alguna información pertinente desde la última consulta realizada. Presupone la programación de una petición AJAX al servidor que tendrá lugar cada cierto intervalo de tiempo.
    - Opcionalmente esos resultados producirán una actualización del DOM, presentando la información pertinente al usuario.

- **Solicitud periódica de intervalo largo (*Long Polling*)**: En este modelo, el servidor mantiene las peticiones recibidas en una cola de procesamiento, hasta que hay alguna información que devolver a los clientes. Por su parte, los clientes vuelven a enviar nuevas peticiones, cuando la anterior se ha recibido correctamente.



Polling periódico

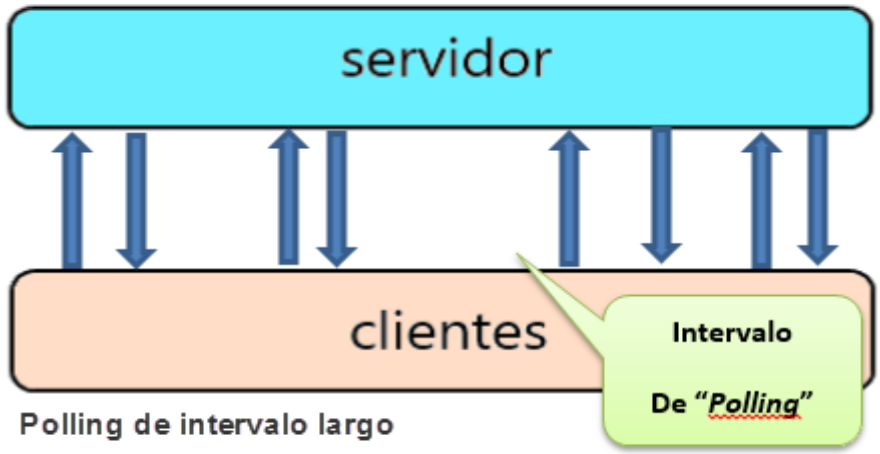


*Fig. 19: Esquemas de "*polling*" a un servidor: Solicitud periódica y solicitud de intervalo largo*

Estas técnicas son sencillas de implementar utilizando JavaScript y solicitudes AJAX (mediante el objeto ***XhrHttpRequest***), pero tienen una serie de desventajas que pretende eliminar esta nueva API, al tiempo que aporta interesantes novedades.

Inconvenientes de las técnicas tradicionales

Por un lado, las comunicaciones TCP ordinarias por puertos diferentes al 80 suelen estar bloqueados por temas de seguridad por los administradores. Por otra parte, la utilización de estas técnicas de *Polling* tiene varios inconvenientes:

- El *Polling periódico* produce un tipo de latencia muy alta.
- El *Polling de intervalo largo*, tiene un modelo de programación muy poco intuitivo.
- Ambos presentan problemas de escalabilidad
- El colapso del ancho de banda es otro de los problemas habituales de este tipo de arquitecturas.
- El soporte entre dominios (*cross-domain*) también está limitado.

A diferencia de los dos anteriores, el esquema de arquitectura propuesto por esta API sería similar al que vemos en la figura siguiente:

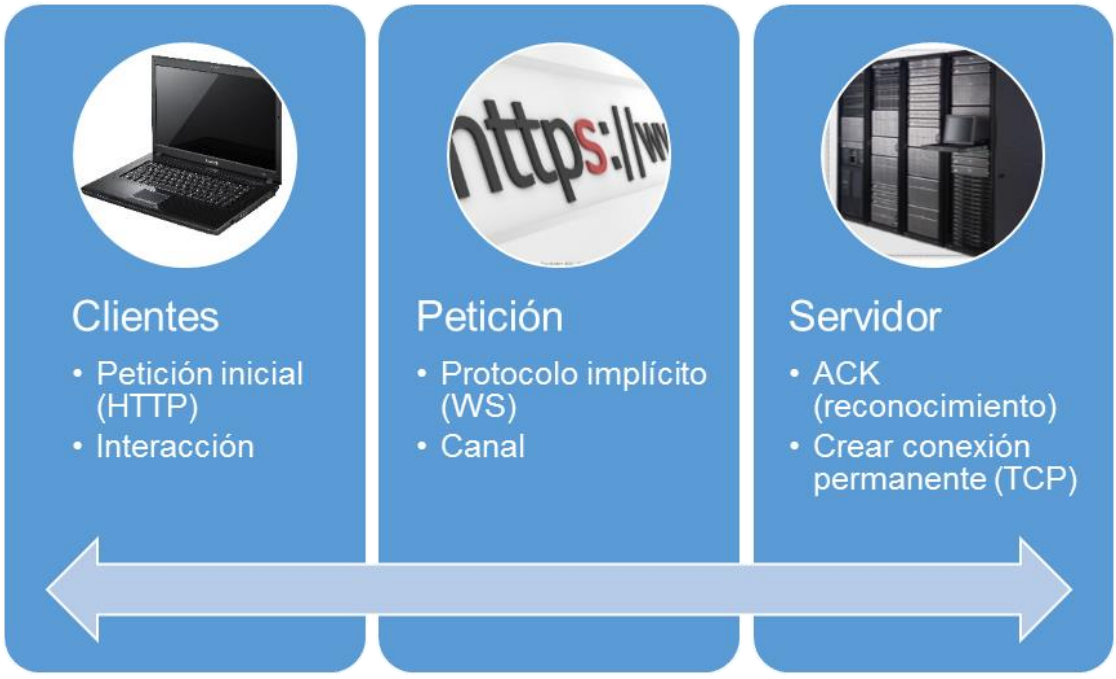


Fig. 20: Esquema de funcionamiento del modelo de WebSockets

Así pues, el uso de WebSockets ofrece una alternativa, suministrando una funcionalidad similar a la apertura de varias conexiones en distintos puertos, pero *multiplexando* diferentes servicios *WebSocket* sobre un único puerto TCP (a costa de una pequeña sobrecarga del protocolo).

Una idea sobre la programación de WebSockets

En la programación de un cliente, una vez creado un objeto de esta clase, deberemos programar la respuesta a los eventos más importantes soportados; a saber: ***onopen, onclose, onmessage*** y ***onerror***.

Según esto, si queremos hacernos una idea de cómo se establecería una de estas conexiones, y como se programaría un proceso de comunicación, podemos concebir algo como esto:

```
// Creamos una variable para almacenar las referencias
// al servicio (obsérvese el protocolo: ws://)
var socket = new WebSocket("ws://dominio/servicio");
```

A continuación programamos una función para recibir las notificaciones del servicio asignándola al evento ***onopen***:

```
// Asignamos el evento onopen
socket.onopen = function() {
 alert("Socket abierto sobre el servicio Ajedrez "on-
line");
}
```

Una vez abierto el servicio, podemos comenzar una conversación sobre el mismo puerto de comunicación. Más adelante, podemos enviar mensajes al servicio mediante el método ***postmessage***:

```
socket.postMessage("Blancas mueven: d4");
```

Y podemos recibir mensajes registrando otro manejador

```
socket.onmessage = function(event) {
       alert("Respuesta de las negras: " + event.data);
       };
```

Donde, como es de suponer, la propiedad ***data*** del objeto ***event*** que se recibe como argumento del evento ***onmessage***, contiene la información de respuesta. El proceso de petición respuesta puede continuar indefinidamente hasta que se cierre la conexión por una de las dos partes o se produzca un error irrecuperable.

Como anticipábamos antes, existe ya algún servicio programado, tal como el servicio "***Echo Test***", de *WebSockets.org*, que, sencillamente, repite de vuelta el mensaje enviado a ese servicio.

Debido al tipo de API implícita en Web Sockets, resulta especialmente útil en depuración la utilización de *Fiddler*, ya que podemos ver todos los datos de intercambio desde el inicio de protocolo de conexión

(*handshaking*).

El soporte adicional de Microsoft

Web Sockets en ASP.NET

Dada la importancia de esta API, Microsoft ha incluido el soporte adecuado en la versión ASP.NET 4.5, como puede leerse en la documentación oficial del MSDN, vinculada al espacio de nombres **System.Web.WebSockets**[104]**.**

Básicamente, este soporte viene dado por 3 clases especializadas. ***AspNetWebSocket***, ***AspNetWebSocketContext*** y ***AspNetWebSocketOptions.***

Su funcionalidad básica es la siguiente:

- ***AspNetWebSocket***: Representa una conexión full-dúplex en tiempo real entre un servidor web y una aplicación cliente ASP.NET
- ***AspNetWebSocketContext***: Suministra una clase base que representa detalles de contexto acerca de una petición individual ***AspNetWebSocket.***
- ***AspNetWebSocketOptions.*** Especifican opciones de configuración para una conexión de Web Socket.

Es de esperar que esta opción popularice muchísimo el uso de esta API en las aplicaciones venideras. Para ver un ejemplo completo en funcionamiento, recomendamos en este momento, ver la demo que incluimos en el apartado de referencias (*"WebSockets in Windows Consumer Preview")*.

SignalR

No obstante, la mayor novedad respecto a esta tecnología por parte de Microsoft y la que está consiguiendo una mayor popularidad en la

[104] http://msdn.microsoft.com/en-us/library/hh160729(v=vs.110).aspx

actualidad es la denominada **SignalR**, una creación de **Damien Edwards**, del equipo de desarrollo de Asp.NET, de la que podemos ver todos los detalles en el sitio http://signalR.net, aunque la información relativa a la documentación está siendo ubicada en el sitio web http://asp.net/signalr. Allí se incluyen vídeos, artículos y código fuente descargable, que ilustra con todo detalle el funcionamiento y cómo podemos poner en marcha soluciones que utilicen WebSockets de forma muy sencilla, con solo hacer referencia a unas librerías y programar la funcionalidad en cliente y servidor en unos pocos pasos.

Utilizando SignalR

Con esta librería podemos añadir cualquier tipo de característica de tipo "tiempo real" a una aplicación ASP.NET del tipo que sea. Cada vez que un usuario recarga una página, o utiliza algún mecanismo de "*Long polling*" para solicitar datos del servidor, esa página es candidata a utilizar SignalR.

Esta tecnología aporta una API para crear llamadas Servidor->Cliente remotas (RPC) que invocan a funciones JavaScript en la página de destino (o cualquier posible plataforma cliente), a partir de un código de servidor escrito en .NET. También incluye una API para manejo de las conexiones (y posibles agrupaciones de éstas, p.ej., usuarios registrados e invitados).

Por tanto se trata de un mecanismo de comunicación "*dúplex*", que deberemos programar teniendo esto en cuenta.

Primer ejemplo

Vamos a ver el uso de estas librerías con un par de ejemplos ilustrativos, comenzando por el más sencillo, que nos ayudará a hacernos una idea del funcionamiento básico desde una aplicación ASP.NET.

Vamos a crear la típica aplicación de "chat", que se basará en la capacidad de enviar mensajes al servidor y que éste replique el mensaje a todos los clientes conectados en ese momento.

Para ello, comenzamos por crear una aplicación ASP.NET vacía, y en el apartado de referencias, utilizaremos NuGet (opción "administrar

paquetes *NuGet*"), para solicitar la descarga e instalación de las librerías necesarias de SignalR.

Una vez instaladas cuando añadimos un nuevo elemento a nuestro proyecto, Visual Studio 2013 *Preview* nos ofrecerá varias posibilidades relacionadas con esta librería como se muestra en la figura adjunta:

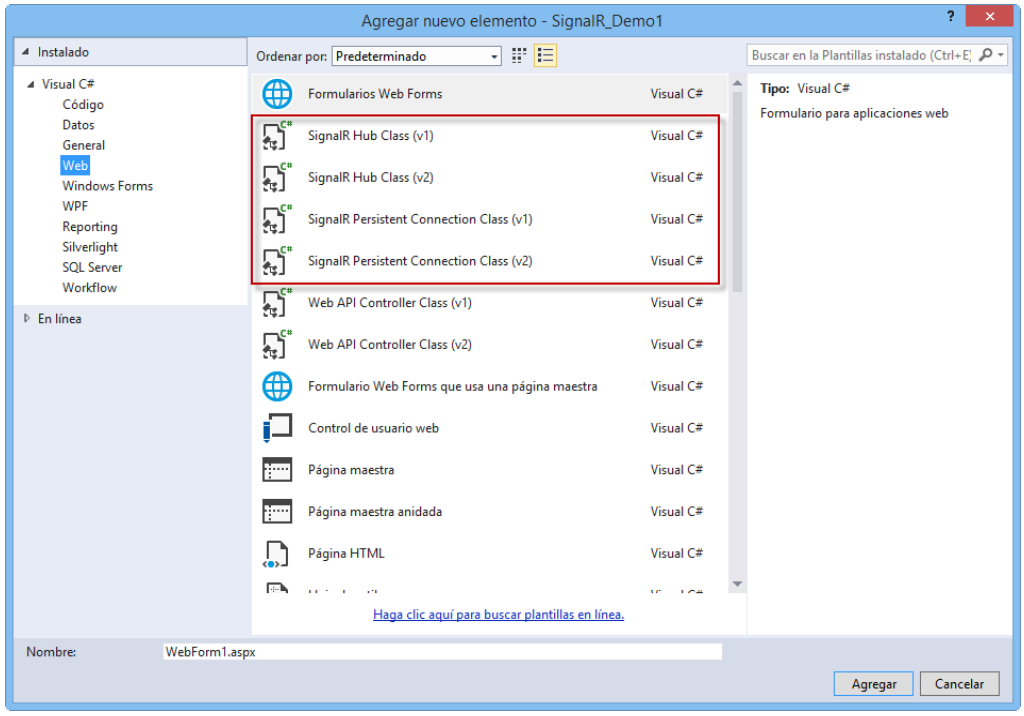


Fig. 21: Elementos SignalR disponibles desde Visual Studio 2013 Preview

Seleccionamos la opción *SignalR Hub Class (v1),* y observaremos que nuestra ventana de soluciones nos muestra, no solo la clase creada, sino las referencias a varias librerías necesarias (las de jQuery y las propias de SignalR), visibles al desplegar la carpeta "Scripts" (ver figura siguiente).

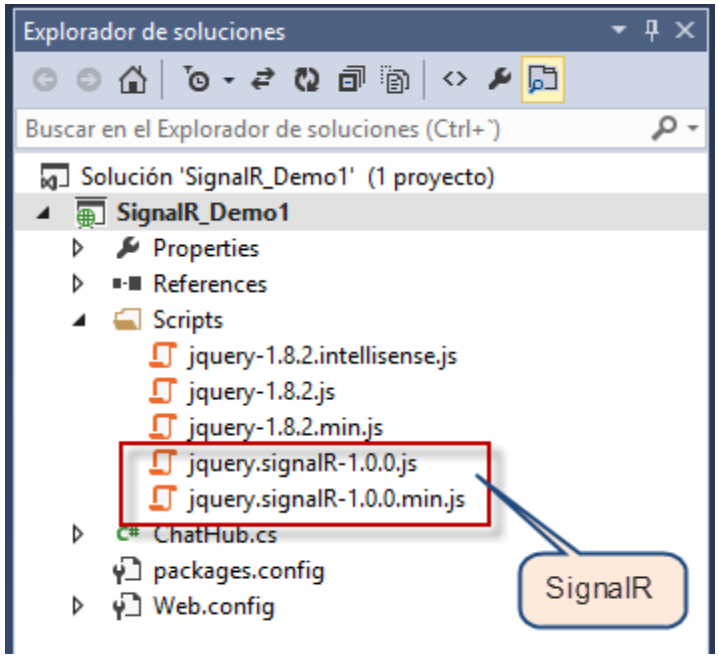


Fig. 22: Librerías de SignalR en la carpeta "Scripts"

Una vez hecho esto, sustituimos el contenido inicial de la plantilla de la clase creada por un método "Enviar", que reciba un par de argumentos: el nombre del que envía el mensaje y el mensaje en sí.

En el cuerpo de la función bastará con una invocación a un método ***enviarMensaje***, que no es parte del objeto ***All***, sino de tipo dinámico (se resolverá en tiempo de ejecución) y que – a su vez- pertenece a uno de los objetos principales para montar un escenario de este tipo: ***Clients***. *Clients* representa a todos los posibles clientes conectados por vía WebSocket a ese servidor en un momento dado. Tendremos pues, el siguiente código:

```
using Microsoft.AspNet.SignalR;

namespace SignalR_Demo1
{
    public class ChatHub : Hub
    {
```

```
        public void Enviar(string nombre, string mensaje)
        {
            // Llamar a broadcastMessage para actualizar
los clientes.
            Clients.All.enviarMensaje(nombre, mensaje);
        }
    }
}
```

La clase ***Hub*** es la clave de todo. Cualquier clase que herede de ella (como nuestra *ChatHub*) dispone de 3 propiedades fundamentales que simplifican todo el proceso de creación del código de servidor: ***Clients*** (que controla el contexto de los clientes conectados), **Context** (que determina de qué cliente es una conexión y sus características), y ***Groups***, que permite hacer un tratamiento diferenciado a grupos de clientes distintos. Además, pone a disposición del programador 3 métodos virtuales que podemos sobrescribir: ***OnConnected***, ***OnDisconnected*** y ***OnReconnected*** (cuyo significado parece bastante evidente y no voy a incluir aquí).

Hay un punto clave, como hemos dicho. La expresión *Clients.All.enviarMensaje* se resuelve en tiempo de ejecución, y por tanto corresponde con el nombre de un método JavaScript, que deberemos escribir en nuestro código cliente (sería el equivalente a un contrato implícito).

Por último, antes de crear nuestro código cliente, tenemos que configurar nuestro servidor añadiendo una instancia de *Global.asax*, para registrar las rutas predeterminadas de nuestro *Hub*, añadiéndola al método *Application_Start,* con lo que tendremos el método siguiente (recuerde que deberá referenciar el espacio de nombres *System.Web.Routing*, donde se encuentra la clase estática *RouteTable*):

```
protected void Application_Start(object sender, EventArgs
e)
{
// Registrar la ruta de hubs predeterminada:
~/signalr/hubs
```

```
    RouteTable.Routes.MapHubs();
}
```

Y eso es todo en la parte del servidor. A continuación, ya estamos en condiciones de crear nuestro código de cliente, y la utilización de jQuery nos facilitará y simplificará mucho el código, como vamos a ver a continuación.

Primero añadimos una página HTML a nuestro proyecto (aquí la llamamos *ClienteChat.html*) y añadimos el código siguiente en la etiqueta ***body***:

```
<div class="Contenedor">
    <input type="text" id="Mensaje" />
    <input type="button" id="enviarMensaje"
value="Enviar" />
    <input type="hidden" id="mostrarNombre" />
    <ul id="discusion"></ul>
</div>
<!--Scripts -->
<!--Referencia a jQuery-->
<script src="/Scripts/jquery-1.8.2.min.js"></script>
<!--Referencia a SignalR -->
<script src="/Scripts/jquery.signalR-1.0.0.js"></script>
<!--Referencia al script SignalR hub autogenerado. -->
<script src="/signalr/hubs"></script>
<!--Proceso que actualiza la página y envía Mensajes.-->
<script type="text/javascript">
    $(function () {
        // Declara un proxy para referenciar el hub.
        var chat = $.connection.chatHub;
        // Crea una funcion que el Hub pueda llamar para
enviar Mensajes.
        chat.client.enviarMensaje = function (nombre,
mensaje) {
            // Crea elementos del DOM para el nombre y el
Mensaje.
            var etiquetaNombre = $('<div
/>').text(nombre).html();
```

```
            var etiquetaMensaje = $('<div
/>').text(mensaje).html();
            // Añade el Mensaje a la página.
            $('#discusion').append('<li><strong>' +
etiquetaNombre
                + '</strong>:  ' +
etiquetaMensaje + '</li>');
        };
        // Solicita el nombre del usuario para añadirlo a
los Mensajes.
        $('#mostrarNombre').val(prompt('Nombre:', ''));
        // Sitúa el foco en la caja de entrada.
        $('#Mensaje').focus();
        // Inicia la conexión.
        $.connection.hub.start().done(function () {
            $('#enviarMensaje').click(function () {
                // Llama al método "Enviar" del hub.
                // (Nótese que el nombre va en
minúsculas)

chat.server.enviar($('#mostrarNombre').val(),
$('#Mensaje').val());
                // Limpia la caja de texto y reubica el
foco.
                $('#Mensaje').val('').focus();
            });
        });
    });
</script>
```

Una vez que iniciemos el servidor abriendo la página con cualquier navegador, estaremos en condiciones de intercambiar mensajes abriendo la misma página con otros navegadores y enviando mensajes que serán inmediatamente actualizados en todos los clientes conectados.

La salida sería algo similar a la de la figura siguiente, en que simulamos un "chat" usando IE11, Chrome y FireFox:

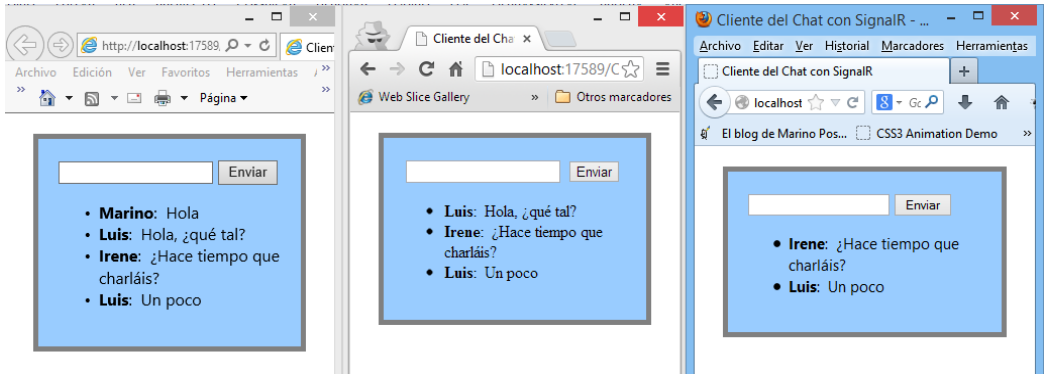


Fig. 23: Salida del código anterior, mostrando a 3 clientes del Hub activos simultáneamente (cada uno en un navegador distinto).

Es importante notar que existe una referencia en el código cliente a una librería que no existe hasta que se pone en marcha el proceso, porque es creada en tiempo de ejecución por SignalR: ***hubs***. Se trata del proxy de comunicación que permite la conexión entre ambos y que situamos como última referencia de los ficheros externos.

Además, la llamada a **`$.connection.hub.start().done(…)`** nos garantiza que primero se llama al servidor, y –cuando éste ha atendido la petición- se programa lo que debe hacer a continuación (una característica típica de jQuery, muy conveniente en este caso).

Basta con ver el apartado "Documentos de Script" en el Explorador de Soluciones", para observar lo que se está utilizando en el momento en que el servidor está listo (el lector puede marcar ese archivo para ver el código fuente generado, mientras está en tiempo de ejecución), como vemos en la figura siguiente:

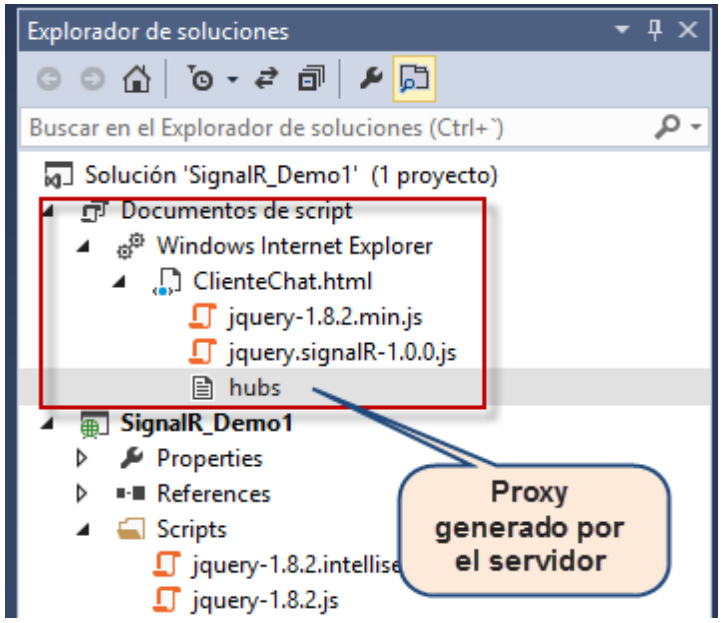


Fig. 24: Archivos en ejecución mostrando el proxy generado por el servidor.

Como vemos, una programación muy sencilla y con muchísimas posibilidades en ámbitos de todo tipo: Juegos por Internet, aplicaciones que se actualizan periódicamente, comunicaciones entre grupos corporativos, notificaciones a clientes, etc.

El ejemplo que hemos visto es una adaptación de dos de los ejemplos oficiales que aparecen en la página "oficial" de *SignalR*, diseñados por **Patrick Fletcher**[105].

Para analizar un ejemplo más complejo, sugerimos que se baje uno de los ejemplos "on-line", accesibles desde la opción "Extensiones y Actualizaciones" de Visual Studio, como por ejemplo, "*Server Broadcast with ASP.NET SignalR (C#)*", que simula una situación de emisión múltiple de un servidor que envía variaciones aleatorias de valores de bolsa. El ejemplo se encuentra completamente explicado en la página http://www.asp.net/signalr/overview/getting-started/tutorial-server-

[105] http://www.asp.net/signalr/overview/getting-started/tutorial-signalr-self-host

broadcast-with-aspnet-signalr.

WebGL: una API fuera del estándar

Lo primero que hay que decir es que WebGL no es un estándar, y no pertenece al grupo de API comentadas en esta obra, tratándose de una iniciativa privada que –eso sí- está contando con el soporte de 3 de los principales protagonistas de la industria: Microsoft (a partir de IE11), Chrome y FireFox.

Wikipedia define **WebGL** en los siguientes términos: **"**una especificación estándar que está siendo desarrollada actualmente para mostrar gráficos en 3D en navegadores web. WebGL permite mostrar gráficos en 3D acelerados por hardware (GPU) en páginas web, sin la necesidad de *plug-ins* en cualquier plataforma que soporte **OpenGL 2.0** (***Open Graphics Library***)[106] u **OpenGL ES** 2.0[107]. Técnicamente es un API para JavaScript que permite usar la implementación nativa de OpenGL ES 2.0 que será incorporada en los navegadores. WebGL está gestionado por el consorcio de tecnología sin ánimo de lucro **Khronos Group**[108].

Se trata de un desarrollo que nace a partir de experimentos realizados con Canvas 3D por **Vladimir Vukićević** en Mozilla. Más adelante (en 2009), al formarse el Grupo Kronos, se unificaron las implementaciones originales, que datan de 2006. La especificación WebGL se basa en la OpenGL ES, y proporciona una API para gráficos 3D. Se utiliza el elemento **<canvas>** de HTML5 (aunque se puede utilizar otros elementos, como SVG) y se accede mediante interfaces DOM. La gestión de memoria es automática y se proporciona como parte del lenguaje JavaScript.

Soporte de los navegadores

A partir de la versión IE11, ya está soportada también por este navegador,

---

[106] http://es.wikipedia.org/wiki/OpenGL

[107] http://es.wikipedia.org/wiki/OpenGL_ES (*OpenGL for Embedded Systems*)

[108] http://es.wikipedia.org/wiki/Khronos_Group

por lo que es de esperar que la adopción crezca rápidamente a medida que se implante el uso de este. El soporte de Firefox y Chrome ya viene de versiones anteriores y Opera dispone de un soporte limitado. Safari lo soporta en sus versiones recientes para OS X.

Siempre según Wikipedia, "Como WebGL es una tecnología diseñada para trabajar directamente con la GPU (unidad de procesamiento gráfico) es difícil de codificar en comparación con otros estándares web más accesibles, y es por eso que muchas bibliotecas de JavaScript han surgido para resolver este problema. De todas las existentes, la que cuenta con mayor aceptación y usuarios es **Tree.js**, una biblioteca creada y publicada en GitHub por el español Ricardo Cabello (conocido por su seudónimo de Mr. Doob) en abril de 2010.

El código de esta biblioteca habría sido primeramente desarrollado en **ActionScript** y luego traducido al JavaScript. Los dos puntos decisivos para la transferencia a JavaScript fueron no tener que compilar el código antes de cada carga y la independencia de plataforma. Las contribuciones de Cabello incluyen el diseño de la API, **CanvasRenderer**, **SVGRenderer** y ser responsable de la fusión del trabajo de los diversos colaboradores en el proyecto."

Si el lector tiene curiosidad por ver algunos de los ejemplos "oficiales" del funcionamiento de ***Three.js***, puede ver un buen número de ellos en su página web: http://stemkoski.github.io/Three.js/.

Para hacernos una idea de la complejidad de la programación WebGL (y –en general-, en 3D), tenga en cuenta que para programar un escenario 3D se requiere una emulación del contexto real, y eso implica una serie de conceptos que son exclusivos de estos entornos: el escenario, la cámara (el "ojo" que ve la escena que estamos dibujando), el "*renderer*" (o intérprete visual), los manejadores de eventos que permitirán el desplazamiento de la cámara en el espacio, la configuración de las geometrías básicas utilizadas para dibujar, la iluminación (sin la cual no veríamos absolutamente nada), las texturas aplicadas a las superficies definidas que aportarán "solidez" a nuestros dibujos, "*shaders*", "*meshes*", efectos especiales (como la niebla), etc.

Alcance de WebGL y otras alternativas

Por tanto, hablamos de una API que maneja a bajo nivel los recursos gráficos de dibujo, y cuyo manejo precisa un conocimiento de los recursos de diseño en 3D para conseguir incluso resultados básicos, si solo utilizamos JavaScript. Esto supone que la alternativa del uso de WebGL se focaliza en ciertos entornos para los que no existe una alternativa de calidad y donde se precisa todo lo que un entorno de 3D puede ofrecer, como fragmentos publicitarios que simulen espacios tridimensionales, juegos que exigen "*renderización*" de texturas de calidad, emulación de superficies y espacios virtuales, etc.

Babylon

Microsoft, consciente de este problema, ha creado una librería que pretende solventar parte de estos problemas y facilitar la programación, llamada *Babylon*.

Con su uso se facilita mucho la implantación de contextos 3D sin tener que bajar al detalle de cada aspecto requerido por este tipo de programación. El sitio oficial, que contiene un paquete de ejemplos bastante espectacular, se encuentra en http://www.babylonjs.com/ y la librería está disponible en ***GitHub*** en la dirección https://github.com/BabylonJS/Babylon.js. La figura siguiente muestra uno de los ejemplos conseguidos mediante esta librería:

Fig. 25: Demo de WbGL en el sitio web oficial de esta librería

Programando un escenario básico en WebGL con Babylon

Afortunadamente, con el uso de esta librería podemos mostrar una primera aproximación a la programación WebGL sin necesidad de entrar en los entresijos del desarrollo 3D más de lo imprescindible.

Comenzamos por un sencillo proyecto Web vacío en el que creamos una página que tenga un elemento **<canvas>** que servirá de superficie de interpretación gráfica. El código HTML necesario sería algo como esto:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=utf-8"/>
    <title>Demos de Web GL- Demo1</title>
    <style>
        html, body {
            width: 100%;
            height: 100%;
            padding: 0;
            margin: 0;
            overflow: hidden;
        }
        #CanvasWebGL {
            width: 100%;
            height: 100%;
        }
    </style>
    <script src="Scripts/babylon.1.2.1.js"></script>
</head>
<body>
    <canvas id="CanvasWebGL"></canvas>
</body>
</html>
```

Como vemos, se trata de un solo elemento **<canvas>** con un nombre, que nos servirá más adelante para referirnos a él desde el código de *script*. También hemos incluido unos valores básicos de estilo, tanto para la página como para el *canvas*.

A continuación del elemento *canvas*, incluimos un *script* que es el que pondrá en marcha todo el proceso:

```
<script>
    if (BABYLON.Engine.isSupported()) {
        var lienzo =
document.getElementById("CanvasWebGL");
        var motor3D = new BABYLON.Engine(lienzo, true);
    }
    // usamos el motor anterior para crear la escena
    var escena = new BABYLON.Scene(motor3D);
    // La escena es un contenedor de entidades, como
    // la cámara, las luces y los objetos a dibujar
    var camara = new BABYLON.FreeCamera("Camara", new
BABYLON.Vector3(4, -1, -10), escena);
    var luz0 = new BABYLON.PointLight("Lateral", new
BABYLON.Vector3(250, 100, 100), escena);
    var esfera = BABYLON.Mesh.CreateSphere("Esfera", 16,
3, escena);
    // Finalmente, tenemos que crear un bucle de
    // interpretación visual (Render loop)
    var bucleVisual = function () {
        // Creamos un nuevo marco de dibujo (frame)
        motor3D.beginFrame();
        escena.render();
        // Presentar
        motor3D.endFrame();
        // Registro del frame
        BABYLON.Tools.QueueNewFrame(bucleVisual);
    };

    BABYLON.Tools.QueueNewFrame(bucleVisual);
</script>
```

Como puede ver el lector, comprobamos primero el soporte del motor WebGL, y utilizamos el elemento *canvas* para crear una instancia del motor 3D (*BABYLON.Engine*), pasándole nuestro *canvas* como parámetro.

A continuación, nos serviremos de los métodos del motor de Babylon, para ir creando los elementos imprescindibles para poder representar algo en ese contexto: la escena (el contexto), la cámara (el "ojo"), la luz (que ilumina la escena desde uno o más puntos, ya que podemos definir tantos como queramos), y el objeto que incluimos en la escena (en este caso, una esfera).

Como en este primer ejemplo no hay animación de ninguna clase, solo vamos a crear una instantánea o marco visual que interpretar (un *frame*). Eventualmente, podríamos crear dinámicamente tantos como queramos, por ejemplo como respuesta a eventos de ratón o teclado.

Para hacer esto, la función anónima que lo carga es asignada a la variable ***bucleVisual*** de forma que podamos pasar ese "mecanismo de construcción de marcos" al motor 3D. La función crea un marco, lo interpreta visualmente (ahí es donde interviene el trabajo pesado de la GPU gracias al motor WebGL), y –una vez terminado el proceso- lo registra para su uso mediante una llamada a una de las herramientas de BABYLON, (***QueueNewFrame***) encargada de poner en cola de proceso los marcos que le van llegando. Finalmente, volvemos a llamar a este método desde fuera de la función para poner en marcha todo el proceso.

El resultado visual mostrado en IE11 es el de la figura xx (los otros navegadores ofrecen idéntico resultado, por supuesto):

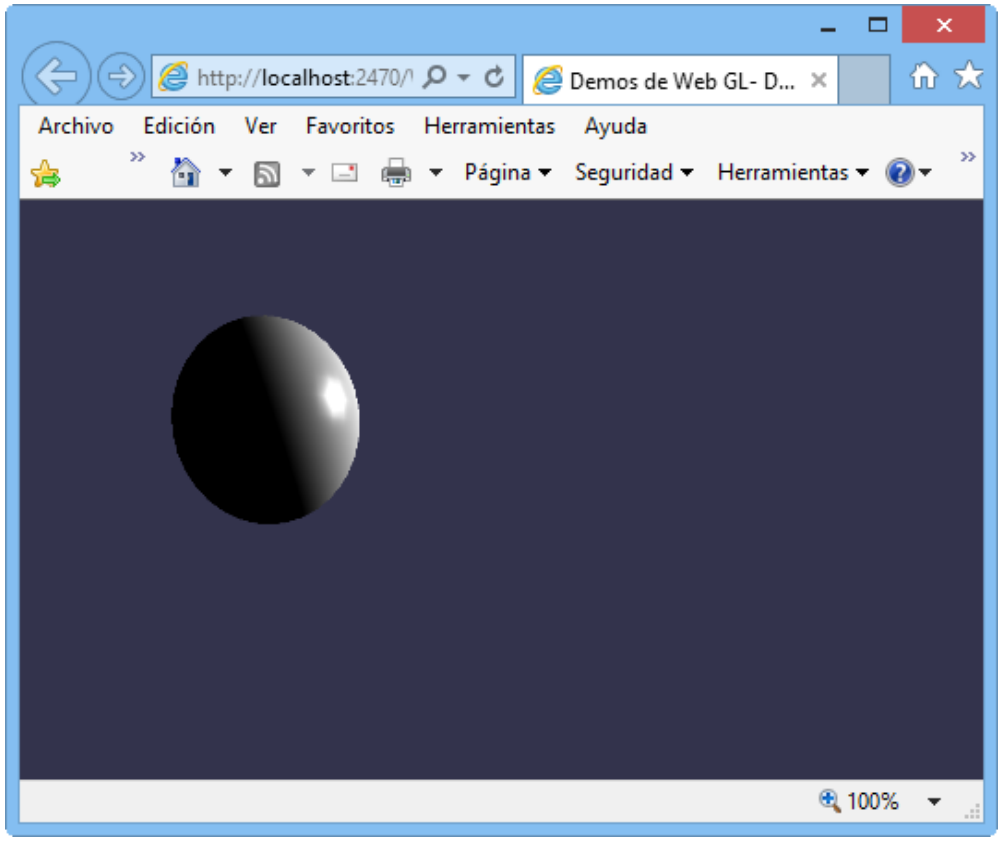


Fig. 26: Salida del código anterior en IE11

Una de las formas sencillas de incluir animaciones es añadir un método de proceso a la propiedad ***beforeRender*** de la escena. Por ejemplo, podemos producir una rotación de la esfera añadiendo al *script* anterior (antes de la función ***bucleVisual***), el código siguiente:

```
// Función de animacion
    var alpha = 0;
    esfera.scaling.x = 0.33;
    esfera.scaling.z = 1.25;
    escena.beforeRender = function () {
        esfera.rotation.x = alpha;
        esfera.rotation.y = alpha;

        alpha += 0.01;
```

```
    };
```

Con esto, estamos modificando la escala de la esfera, tanto en el eje de las X (a un tercio del original), como en el eje de las Y (un 25% más que el original), de forma que –al rotar- nos ofrezca distintas perspectivas visuales y poder apreciar el movimiento. Si el lector reproduce esto, podrá observar que la luz ("*Lateral*"), provoca al rotar que la superficie iluminada sea muy distinta según se progresa en la rotación, como muestran las figuras siguientes:

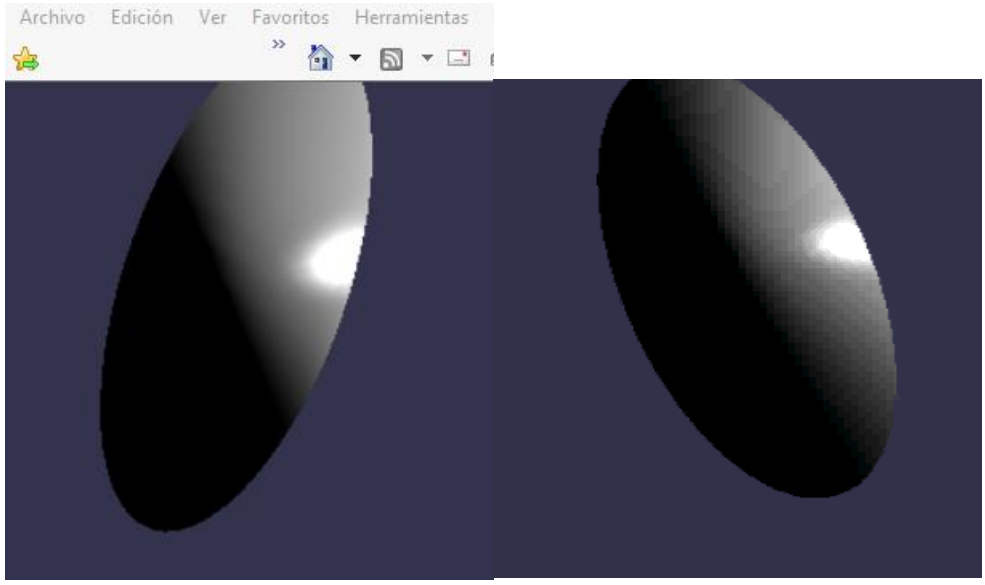


Fig. 27 y 28: La esfera anterior, deformada y sometida a un movimiento de rotación

Como podemos ver al ejecutar este código, la esfera solo tiene definido un tono de color (gris) y las variaciones observadas son debidas a la presencia del único foco de luz definido. Tanto en este caso, como en la mayoría de casos prácticos, lo normal es que deseemos "pintar" las superficies definidas (genéricamente se les denomina mallas) con texturas o superficies predefinidas que puedan aplicarse a la geometría definida aportándole un aspecto más realista.

Esto se consigue con el uso de Materiales (*Materials*), que definen como

una superficie dibujada aparece finalmente. En Babylon, existe un objeto llamada ***StandardMaterial*** que aporta propiedades muy interesantes para conseguir efectos relacionados con las texturas:

- **diffuseColor** y **diffuseTexture**: Definen el color base de la malla.
- **ambientColor** y **ambientTexture**: Definen el color ambiente de la malla (se puede utilizar para los mapas de luz, por ejemplo).
- **specularColor** y **specularTexture**: Definen el color especular de la malla.
- **emissiveColor** y **emissiveTexture**: Definen el color emitido por la malla (el color que el objeto tiene sin luz).
- **opacityTexture**: Define la transparencia de la malla.
- **reflectionTexture**: Define el color de reflexión recibida por la malla (puede ser una textura o un espejo dinámico tipo *Sourire*).
- **bumpTexture**: Define el nivel de la protuberancia de la malla sobre una base "*per-pixel*". Se debe proporcionar una imagen tipo "*bitmap*".
- **alpha**: Define la transparencia global de la malla.

En general, debemos de tener en cuenta que, mientras las propiedades de color definen colores simples, las propiedades de textura utilizan un "*bitmap*".

De acuerdo con esto, podemos utilizar cualquier gráfico y aplicarle su contenido a un objeto tipo "*material*", configurar alguna de las propiedades definidas más arriba y utilizarlo con nuestra esfera para cambiar su aspecto. Nos bastaría con añadir las siguientes líneas (después de la definición de la esfera) para aplicar estas propiedades a nuestro objeto.

```
// Material
var material = new BABYLON.StandardMaterial("inicial",
escena);
material.diffuseTexture = new
BABYLON.Texture("Danysoft_multiple2.png", escena);
material.emissiveColor = new BABYLON.Color3(0.3, 0.3,
0.3);
```

```
esfera.material = material;
```

Y hecho esto, obtendríamos la misma estructura funcional que antes, pero "pintaríamos" nuestra esfera con una versión en mosaico del logo de ***Danysoft***, con siguiente resultado visual:

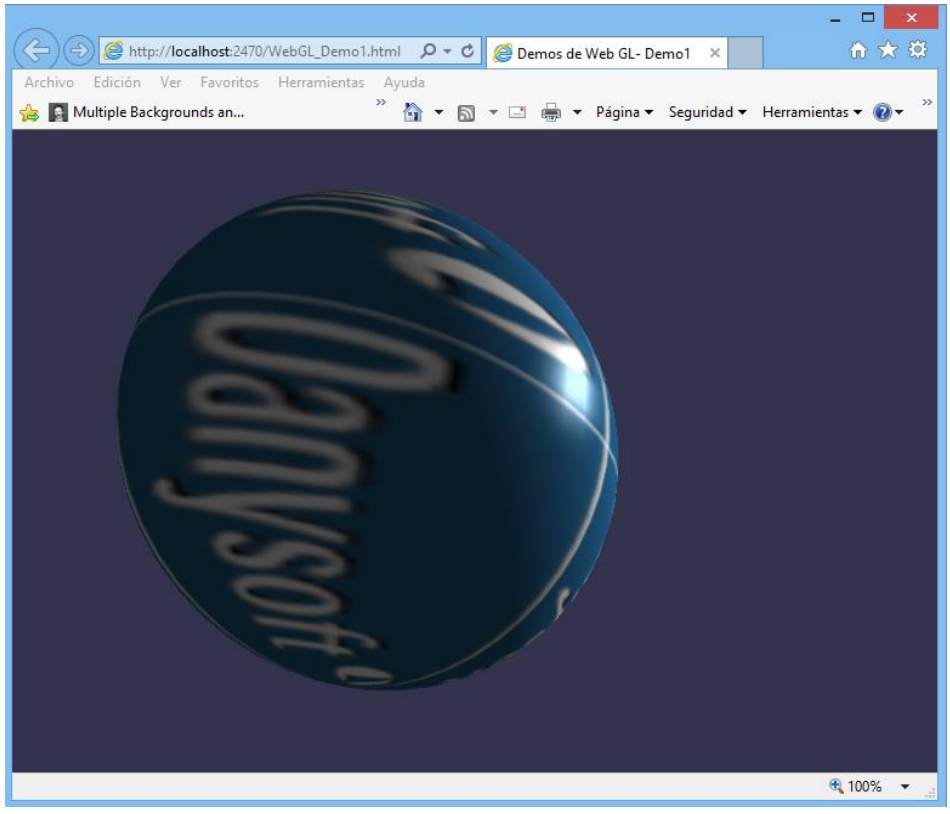


Fig. 29: Salida de la esfera inicial con las texturas aplicadas.

Y son muchas más las posibilidades que esta librería ofrece para hacer nuestras primeras pruebas con WebGL. Recomendamos al lector interesado visitar el sitio Web oficial para una descripción detallada de la documentación, que –desde luego- se sale del ámbito de esta obra.

Espero, no obstante, que este sencillo ejemplo, sirva de test inicial sobre el que poder hacerse una idea del funcionamiento, en caso de animarse a incorporar este API en los desarrollos habituales o sirva como punto de

partida para proyectos más ambiciosos en publicidad, juegos, simulaciones, etc.

Referencias

- Modern.IE: http://www.modern.ie/es-es
- "*Unifying touch and mouse: how Pointer Events will make cross-browsers touch support easy*": http://bit.ly/XqiqcD
- Especificación Drag&Drop: http://www.w3.org/TR/2011/WD-html5-20110113/author/dnd.html
- "*Objetos IndexedDB*", Microsoft MSDN, http://msdn.microsoft.com/es-es/library/ie/hh920759(v=vs.85).aspx
- "*Una sencilla lista de tareas pendientes utilizando IndexedDB de HTML5*", **Paul Kinlan** en *HTML5Rocks*, Dic. 2010, http://www.html5rocks.com/es/tutorials/indexeddb/todo/
- Web Sockets en MSDN Microsoft: http://msdn.microsoft.com/es-es/library/ie/hh673567(v=vs.85).aspx
- "*WebSockets in Windows Consumer Preview*", IEBlog, http://blogs.msdn.com/b/ie/archive/2012/03/19/websockets-in-windows-consumer-preview.aspx
- "*FleepBook Demo*" (Web Sockets): https://websockets.interop.msftlabs.com/flipbook/
- Lista completa de *Polyfills* y *Shims*: https://github.com/Modernizr/Modernizr/wiki/HTML5-Cross-Browser-Polyfills

Índice

Sitios Web relacionados

A continuación te incluimos enlaces que pueden ser de tu interés:

Adquisición de licencias de Visual Studio:

http://www.danysoft.com/shop

Servicios de formación y/o consultoría:

http://www.danysoft.com/servicios

Videos sobre Visual Studio:

http://www.danysoft.com/secciones/comunidad/videos
http://www.youtube.com/danysoftech

Eventos y seminarios Web:

http://www.danysoft.com/secciones/danysoft/eventos-danysoft

Libros Danysoft sobre Visual Studio:

http://www.danysoft.com/productos/libros-danysoft.html

Visual Studio 2010:
http://msdn.microsoft.com/en-us/library/dd831853.aspx