

HTML5
CSS3 y JavaScript

Marino Posadas

netalia
just for developers

HTML5
CSS3 y JavaScript

Marino Posadas

ADVERTENCIA LEGAL

Todos los derechos de esta obra están reservados a Marino Posadas y netalia.

El editor prohibe cualquier tipo de fijación, reproducción, transformación o distribución de esta obra, ya sea mediante venta, alquiler o cualquier otra forma de cesión de uso o comunicación pública de la misma, total o parcialmente, por cualquier sistema o en cualquier soporte, ya sea por fotocopia, medio mecánico o electrónico, incluido el tratamiento informático de la misma, en cualquier lugar del mundo.

La vulneración de cualesquiera de estos derechos podrá ser considerada como una actividad penal tipificada en los artículos 270 y siguientes del Código Penal.

La protección de esta obra se extiende al mundo entero, de acuerdo con las leyes y convenios internacionales.

© Marino Posadas, 2012
© netalia, S.L. 2012

HTML5, CSS 3 y JavaScript

Autor: Marino Posadas
Responsable editorial: Paco Marín
Diseño de cubierta: Silvia Gil
Maquetación: Silvia Gil

Editado por netalia
http://www.netalia.es - http://www.dnmplus.net
@netalia_es

Editado en formato digital e impreso.

Impreso
Precio: 29,50€
ISBN: 978-84-939296-2-6
Edición impresa en Navarra por Ulzama

Digital
Precio: 9,95€
ISBN: 978-84-939296-1-9

A Juan José García y Javier García,
con la promesa de que jamás les obligaré a leer nada mío.

A mis sobrinos

A Milagros… siempre.

«Hay solo dos clases de lenguajes de programación: aquellos de los que la gente está siempre quejándose y aquellos que nadie usa»
Bjarne Stroustrup

«Los proveedores de software están intentando hacer sus productos más amigables para el usuario. Su mejor aproximación hasta el momento ha sido tomar sus antiguos folletos y estampar las palabras 'amigable para el usuario' en la portada»

Bill Gates

Agradecimientos

A Paco Marín, de Netalia, por animarme siempre en estas empresas.

A mis compañeros del grupo de MVP de Microsoft en España, con Cristina González Herrero a la cabeza, y los componentes del DPE y divulgación en Microsoft: Alfonso Rodríguez, José Bonnín, David Carmona, David Salgado y todos los demás.

A Paul Cotton de la W3C, por sus palabras y ánimos en la entrevista que nos concedió en su visita a Madrid.

A mis seguidores de Twitter, Facebook y otras redes sociales. Siempre son una motivación para continuar la tarea.

Y a las empresas colaboradoras y amigas o que han mostrado interés en este proyecto, como Alhambra-Eidos, Aula MCT, CAS Training, DanySoft, MegaTraining, Ceticsa, MSL, Desfufor, GadeSoft. ¡Que sigamos mucho tiempo en la brecha!

índice

1. Introducción 13

Objetivos de la W3C 14
Objetivos de esta obra 14
Las Herramie˜ tas 14
HTML 5: La ˜ ueva propuesta 15
El World Wide Web Co˜ sortium 16
Está˜ dares, HTML y la W3C 16
El lapso de tiempo desde el último está˜ dar 17
El problema de las fechas de termi˜ ació˜ 18
Los ˜ avegadores y el está˜ dar: IE10 como objetivo 19
Nuevas reglas del juego 20
Los ˜ avegadores a˜ tiguos 21
Los objetivos del está˜ dar e˜ la práctica 22
Motores de JavaScript 23
Librerías y Le˜ guajes que compila˜ a JavaScript 24
¿Por qué ˜ o esperar a que el está˜ dar esté termi˜ ado? 25
El sueño de u˜ a Web Semá˜ tica 26
Los Ide˜ tificadores u˜ iformes de recursos (URI vs. URL) 27
URI y Semá˜ tica 27
Especificacio˜ es y pruebas de laboratorio 29
Los test 29
Los sitios alter˜ ativos de pruebas de co˜ formidad 30
Los sitios oficiales 31
Las aplicacio˜ es web y el ˜ uevo modelo de aplicacio˜ es e˜ Wi˜ dows 8 32
´ plicacio˜ es Wi˜ dows 8 33
El ˜ uevo modelo de aplicacio˜ es Wi˜ dows 8 34
La posició˜ de Silverlight y Flash e˜ Wi˜ dows 8 35
Wi˜ JS y los co˜ troles "de fábrica" 35
La divisió˜ gra˜ ular de JavaScript 38

Ejecució˜ y soporte del está˜dar 39
Los motores Chakra y los dos co˜textos de ejecució˜ 40
Habla˜do sobre el está˜dar: la opi˜ió˜ de u˜ protago˜ista 41
E˜trevista co˜ Paul Cotto˜ 41
Refere˜cias 52
Refere˜cias de la Web Semá˜tica 52

2. Herramientas y depuración 55

Las herramie˜tas de los ˜avegadores 55
Herramie˜tas y depuració˜ 55
La herramie˜ta de desarrollo de I˜ter˜et Explorer (F12) 56
Fiddler 59
Versio˜es de Fiddler y herramie˜tas añadidas 60
´rquitectura de Fiddler 62
Comparativa de capacidades de análisis de tráfico de red entre F12/IE y Fiddler 63
Generación de informes 64
Otras extensiones de Fiddler 66
Fiddler como complemento de navegadores 66
FireBug para FireFox 67
Vistas 3D de cualquier página 69
Las herramientas de desarrolladores de Google Chrome 70
Opera 72
Safari 74
El soporte de HTML 5 en Visual Studio 75
El nuevo soporte de HTML 5 en Visual Studio 2012 76
Page Inspector 78
Soporte de Hojas de Estilo 80
Soporte de JavaScript 84
Microsoft Expression Blend para Visual Studio 2012 87
Plantillas disponibles 88
Referencias 93

3. HTML 5: nuevas etiquetas 95

El problema de la WH´ TWG 95
El marco de trabajo y los objetivos 96
Compatibilidad hacia atrás 97
Sintaxis general 98
El tipo de documento: DOCTYPE 99
Codificación de Caracteres (Encoding) 100
HTML5: Los nuevos elementos 101
Cambios genéricos para todos los elementos: ´ tributos globales 103
Las nuevas etiquetas, por categorías 105
Etiquetas estructurales o semánticas 105
Etiqueta <Section> 106
<section> y la noción de esquema de un documento 108
Etiqueta <article> 109
Etiqueta <aside> 109
Etiqueta <header> 110
Etiqueta <footer> 111
Etiqueta <nav> 112
Implementación de una entrada de un blog 114
Elemento <figure> 116
Nota sobre este conjunto de etiquetas semánticas 118
Multimedia: Elementos <video>, <audio>, <source> y <track> 118
Códec y formatos soportados 118
Utilización de las etiquetas <video> y <audio> 120
Elemento <source> 121
El fragmento de código para video en Visual Studio 2012 121
Elemento <track> 122
Elemento <audio> 124
Elemento <embed> 126
Elemento <mark> 127
Elemento <progress> 127
Elemento <meter> 129
Elemento <time> 130
Elementos <ruby>, <rt> y <rp> 131
Elemento <bdi> 132

Elemento <wbr> 133
Elemento <command> 133
Elemento <keygen> 134
Elemento <output> 136
Elemento <details> 138
El nuevo editor gráfico de Visual Studio 2012 139
Elemento <dataList> 140
Los elementos de trazado gráfico: <canvas> y <svg> 141
El elemento <canvas> 142
Soporte en Visual Studio 2012 143
Dibujando sobre el canvas 144
Rellenos y otras composiciones 146
Dibujando otras figuras básicas 147
Utilizando imágenes en un <canvas> 153
Transformaciones 155
´ ˜imacio˜es y co˜te˜ido di˜ámico 156
El eleme˜to <svg> 158
Soporte de Visual Studio 2012 162
Co˜clusió˜ 166
Refere˜cias 167

4. HTML 5: nuevos atributos 169

La etiqueta <i˜put> y el ´ PI HTML Forms 169
´ tributos de <i˜put> 171
Otros atributos compleme˜tarios 175
Marcadores de posició˜ (Placeholders) y el atributo patter˜ 176
´ tributo patter˜ 177
´ tributo autofocus 178
´ tributo autocomplete 179
Los eleme˜tos <datalist> e <i˜put> y el atributo list 180
Más sobre las validacio˜es 181
´ tributo required 182
´ tributos ˜ovalidate y form˜ovalidate 182

Perso˜alizació˜ de los me˜sajes de error y las ´ PI de validación 184
Las ´ PI de Validación 186
La familia form* 191
´ tributos globales 192
´ tributo contenteditable 196
´ tributo hidden 197
´ tributo spellcheck 197
´ tributos draggable y dropzone 198
Los atributos data- 198
El atributo data- y las aplicaciones HTML5 para Windows 8 199
Referencias 202

5. CSS 3 (Hojas de Estilo en Cascada) 203

El estado actual del estándar 203
CSS 3 en Visual Studio 2012 205
Page Inspector 206
Ventajas de usar CSS 208
Ubicación de los estilos y ámbito de influencia 209
Selectores en CSS 3 210
Extensiones CSS de los navegadores 211
Selectores dependientes 212
´ grupación de selectores 213
Selectores por valor de atributos 215
Nuevos selectores de atributos en CSS 3 216
Pseudo-Clases y Pseudo-Elementos 217
Nuevas pseudo-clases 218
Los pseudo-elementos (antiguos y nuevos) 221
Las nuevas definiciones en CSS 3 222
Modificaciones estructurales 222
Colores 224
Fondos (Propiedad Background) 225
Bordes 231
Bordes definidos mediante imágenes 233

Efectos de sombreado 234
Gestió˜ de textos 238
El tipo de letra 238
Especificació˜ de texto seleccio˜able 238
Fue˜tes descargables y formatos 239
U˜idades de medida de las fue˜tes 241
Características especiales para las fue˜tes Ope˜Type 242
El problema del ajuste del tamaño de las fue˜tes 243
Texto co˜ sombra 244
Colum˜as 247
Exclusio˜es 250
El diseño de caja flexible (Flexbox) 253
El diseño de cuadrícula 256
Regio˜es CSS3 260
´daptación a dispositivos 262
Carga condicional 264
Directivas CSS como extensiones de un navegador 265
Transformaciones 267
Propiedad transform 269
Transiciones y ´nimaciones 272
Diferencia entre transiciones y animaciones 273
Transiciones 273
´nimaciones 277
Recursos y referencias adicionales 282

6. JavaScript 5 y las API relacionadas 283

JavaScript y ECM´Script 283
Peculiaridades de JavaScript 284
Novedades en JavaScript 5 287
Matrices Tipadas 287
El modo Strict 289
Otras implicaciones del modo strict 290
Detección del modo strict 291
Novedades Sintácticas 291
Test del modo strict 292

La orientación a objetos de JavaScript 293
Prototipos 293
Sobre escritura de métodos (Overriding) 294
Creación de objetos mediante Object.create 295
Propiedades de ´ cceso (getters y setters) 296
JavaScript 5 y Reflection 297
Funcionamiento de la palabra reservada this 298
Funciones y el operador new 299
Las ´ PI de JavaScript 301
Ejecución asíncrona de scripts 301
´ rrastrar y soltar (Drag & Drop) 302
Geo-Localización 304
Implementación en un par de casos prácticos 306
´ lmacenamiento local (localStorage) y almacenamiento de sesión (sessionStorage) 309
´ lmacenamiento local (LocalStorage) 312
´ lmacenamiento de datos complejos (IndexDB) 313
´ PI para acceso a ficheros locales (File ´ PI) 314
´ PI que implican procesos asíncronos o comunicaciones 317
Llamadas asíncronas a un servicio mediante ´ J´ X y JSON 318
Web Workers: Tareas asíncronas en el cliente 324
Web Sockets 327
El ´ PI History 330
Referencias 333

capítulo 1

Introducción

HTML es el estándar oficial de la Web. Es el lenguaje en que se han construido (directa o indirectamente, mediante herramientas de desarrollo), la inmensa mayoría de los 7.000 millones de páginas que, según las estadísticas, están disponibles hoy día en Internet. Bien, pues HTML está cambiando. Está cambiando el lenguaje de marcas, y los otros estándares vinculados con él, CSS y JavaScript.

Esto no es algo habitual, aunque tampoco es nuevo: la versión en uso por la mayoría de los sitios en la actualidad es HTML 4, si bien nunca existió una versión 1.0. De hecho, la primera versión como tal que se recoge en la entidad de estandarización que se encarga de estas labores para Internet (***World Wide Web Consortium***, o ***W3C***), es la 3.2. Todo lo anterior pertenece a los "oscuros inicios de la Web", donde la guerra de tecnologías trataba de imponerse, las empresas intentaban aprovechar la red en su propio beneficio y no existía ninguna entidad que pusiese orden y concierto en ese caos. `acronym`

Fue —precisamente— el creador de la Web, **Tim Berners Lee**[1] (`http://www.w3.org/People/Berners-Lee`) el que lideró el proyecto inicial de creación de esa entidad, con sede en el M.I.T., que ahora dispone de otras subsedes por todo el mundo, incluida España[2].

Estas sedes mantienen secciones de frecuente actualización para todos los estándares que manejan (varios cientos en realidad), así como de eventos, software y mecanismos de validación de documentos, informática móvil, diseño de aplicaciones y un largo etcétera.

[1] Lee ejerce funciones de director y trabaja en colaboración con otros equipos del resto de organizaciones miembro

[2] Disponible en www.w3c.es

Objetivos de la W3C

Los principios rectores de esta organización están claramente declarados en sus páginas con el lema "Una Web para todo el mundo", o sea: lenguajes y formatos estándar que sean adoptados por todos los fabricantes para fomentar el juego limpio y la universalidad de la Web, de forma que sea accesible de forma "*independiente del dispositivo*, *del hardware*, *de la infraestructura de red*, *idioma*, *cultura*, *localización geográfica*, *o habilidad física o mental*". En suma, una Web universal.

Según sus propias palabras, la W3C propugna, además, una visión basada en 3 principios:

- "Web de los Autores y Consumidores" (todos lo somos en cierta medida)
- "Web de Datos y Servicios" (la compartición y almacenamiento de la información suministrada por los protagonistas anteriores), y la
- "Web de Confianza", que, según ésta entidad, se conseguirá con la adopción completa de los nuevos estándares que incluyen tecnologías que fomentan la confianza y la responsabilidad. La llamada Web Semántica, y los mecanismos que refuerzan la Seguridad y la Privacidad, son cruciales para conseguir esta meta.

Objetivos de esta obra

Explicaremos aquí lo fundamental de estos propósitos de cara al desarrollador, con la idea de proporcionarle un marco de referencia adecuado sobre el que evaluar las novedades y los cambios que han tenido lugar respecto a versiones anteriores y adaptarlos a la nueva situación.

Asumimos, por tanto, que el lector conoce las versiones previas de estas tecnologías (o, al menos, lo fundamental de ellas), y lo que pretende es una puesta al día, al tiempo que dispone de una referencia lo suficientemente amplia como para permitir el abordaje de nuevos proyectos o actualizaciones de otros más antiguos.

Además, dado que la construcción de este estándar está teniendo una gran repercusión en Internet, añadiremos al final de cada capítulo y en la parte final del libro un apartado de referencias por categorías, para facilitar la búsqueda de sitios relacionados y de interés para desarrolladores.

Las Herramientas



Se pueden utilizar las herramientas de desarrollo más conocidas o convenientes en cada caso particular, pero en esta obra utilizaré la última versión disponible de Visual Studio (**Visual Studio 2012 RC**), que se encuentra disponible de forma gratuita en los servidores de Microsoft. En el capítulo segundo, haremos un recorrido por otras herramientas, incluidas las de los propios navegadores que ofrecen un soporte muy rico para la depuración y análisis de sitios y/o páginas. También añadiremos allí las referencias Web correspondientes.

Naturalmente, para el trabajo con estas tecnologías, el lector puede usar igualmente versiones anteriores de Visual Studio (existen complementos de V. Studio 2010/2008 que capacitan para el trabajo con HTML5), u otros entornos de desarrollo, como Eclipse.

Además, todo el código fuente será testado en distintos navegadores y está a disposición de los lectores, tanto en las páginas de distribución de esta obra, como en mi sitio Web (`www.ElAveFenix.net/Libros`).

HTML 5: La nueva propuesta

Tras un período de "silencio, mantenimiento y fracasos", como lo calificaba algún analista, la W3C[3] ha vuelto a la carga con ánimos renovados. Como veremos en este mismo capítulo, **Paul Cotton** (uno de los responsables del estándar, vinculado a Microsoft), declara que "el objetivo es construir algo llamado la *Open Web Platform* (Plataforma Web Abierta)".

Para ello, se tienen que cumplir algunos requisitos. El más importante, conseguir una adopción universal por parte de la empresa y el mundo académico, algo de lo que ya existen precedentes, como sucedió en la creación del estándar XML (en el que todas las aristas del polígono empresarial estaban representadas[4] y también lo estaba el mundo académico). Hoy, XML está aceptado por todos y se usa universalmente. Por todo esto, conviene saber algo sobre la organización que se encarga de estas tareas.

---

[3] http://www.w3.org/

[3] 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/

El World Wide Web Consortium



La W3C se creó en 1994, y su objetivo inicial fue similar al de las entidades ISO o ANSI. Sus miembros actuales son más de 300 organizaciones de todo el mundo que corren con los gastos de financiación[5] —aparte de algunos ingresos estatales— y lo componen profesionales de la informática de todos los sectores, cuya misión es definir y normalizar la utilización de los lenguajes usados en Internet mediante un conjunto de *Recommendations* (recomendaciones) que son publicadas libremente en su sitio Web y aprobadas por comités de expertos compuestos por representantes nominales de la propia W3C y técnicos especializados de las más importantes compañías productoras de software, distribuidoras y centros de investigación. Baste citar entre ellos a Adobe, AOL, Apple, Cisco, IBM, Intel, Lotus, Microsoft, Motorola, Netscape, Fujitsu, Sony, Panasonic, HTC, Novell, Oracle, etc.

Su sitio Web principal lo alberga el prestigioso Massachussets Institute of Technology (M.I.T.) a través de su Laboratorio de Informática (`http://www.lcs.mit.edu/`) en EE.UU., y dispone de réplicas en el INRIA (`http://www.inria.fr`) en (Francia) y la Universidad de Keio (`http://www.keio.ac.jp/`) en Japón. Además existe un conjunto de delegaciones por todo el mundo, a las que hay que añadir otras organizaciones que coordinan actividades relacionadas de carácter abierto[6].

Hasta el momento, han desarrollado más de 20 especificaciones técnicas. Para los interesados en la historia de la Web, recomendamos: "*The World Wide Web History Project*" (`http://1997.webhistory.org/home.html`), donde se conservan, incluso, los "*posts*" y correos originales de los diversos colaboradores de **Tim Berners-Lee**, según se iba desarrollando[7].

Estándares, HTML y la W3C

Quizá quien se aventure por primera vez en este asunto de los estándares de internet, pueda preguntarse exactamente qué se entiende por un estándar. En una obra publicada

[5] La W3C recibe aportaciones de 3 fuentes distintas: Cuotas de los Miembros del W3C, Becas de investigación y otras fuentes de subvención pública y privada, y donaciones individuales de dinero y equipamiento.

[6] 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.

[7] Es curioso, por ejemplo, leer el "post" de Marc Andreessen, sugiriendo una nueva etiqueta HTML que el propuso que tuviera el nombre <IMG>…

por esta misma editorial, **Javier Holguera** define los estándares de la siguiente forma: "*Existen dos tipos de estándares: de facto y de iure. Los primeros suele establecerlos una compañía o producto, que debido a su uso masivo en un campo, se convierten en referencia en el mismo, aceptados por el resto de miembros del mercado; los segundos son coordinados por un organismo de estandarización y aseguran un proceso de revisión y consenso entre las partes interesadas, antes de su publicación. Este proceso de revisión y consenso es, precisamente, el que los hace más beneficiosos que los estándares de facto, que normalmente responden a los intereses de una única parte.*

Existen muchos ejemplos de ambos tipos de estándares, e incluso algunos de ellos que han pasado por ambas fases. Por ejemplo, PDF nació en 1993 y su popularidad lo convirtió en el estándar de facto de documentos imprimibles. Sin embargo, no sería hasta 2005 cuando se convertiría en un estándar de iure, con la publicación de PDF/A en ISO, uno de los organismos de estandarización más importantes."

Como apuntábamos al inicio, la primera especificación relevante publicada por la W3C fue la versión HTML 3.2, ya que su primer trabajo fue un intento de poner orden en la situación anterior con una especificación de compromiso que aclarase de alguna forma el caos existente y que se bautizó como HTML 2.0, entendiéndose que todo el desorden anterior a ese momento recibiría genéricamente el nombre de HTML 1.0, aunque nunca hubiera existido tal especificación.

Un tiempo después, se pudo observar que el trabajo que realizaba W3C para la normalización difería notablemente de los planes de Netscape, por lo que hubo que hacer tabla rasa del trabajo anterior y abordar el problema con seriedad, a partir de la situación real. Al conjunto de dicho trabajo se le llama de forma genérica HTML 3.0 ó HTML+. Finalmente, llegó HTML 3.2, que recogía todas las principales características de Netscape y de Internet Explorer, y que es la primera a la que puede llamársele estándar *de facto*.

El lapso de tiempo desde el último estándar

Han transcurrido más de 11 años desde la última publicación de HTML (la versión 4.01[9], para ser exactos). En el medio, mucho trabajo, pero no tantos estándares generados, a veces por falta de coordinación otras por conflictos de intereses. Lo más reseñable, la especificación XHTML 1.0[10], que —en palabras de la propia W3C— es una refor-

[8] Internet Explorer 9 y HTML5 (Javier Holguera): http://www.netalia.es/libros

[9] La especificación HTML 4.01 data de Diciembre de 1999: http://www.w3.org/TR/1999/REC-html401-19991224/

[10] http://www.w3.org/TR/2002/REC-xhtml1-20020801/

mulación de HTML 4.0 en XML 1.0. O sea, un HTML con sintaxis estricta. Su adopción ha sido insuficiente.

El problema principal estriba en la gran cantidad de autores que ignoran la existencia de estos nuevos estándares, y la enorme base de páginas ya creadas que pedían a gritos seguir funcionando, aunque no cumplieran con las restricciones impuestas. Posteriormente, hubo un nuevo intento, todavía más restrictivo: XHTML 2.0. Este ni siquiera vio la luz como recomendación final, ya que se produjo un claro desacuerdo entre los partidarios de que las páginas que no lo cumpliesen dejasen de funcionar y los más pragmáticos, que abogaban por una "política de hechos consumados". Más de la mitad de las páginas existentes hubiesen dejado de mostrarse correctamente (o incluso, incorrectamente: no se hubieran mostrado en absoluto).

Por fin, hace unos 4 años, se produjo un nuevo intento de unificar las fuerzas en busca de algo nuevo, que no cayese en los errores o problemas anteriores, y que se hiciese eco de los nuevos requisitos que tiene la Web de hoy día. Se reanudaron los contactos, se empezó a detectar un interés de toda la industria en una renovación, y las empresas más importantes del mundo acordaron participar en el nuevo proyecto.

No aburriré al lector con estadísticas detalladas, pero se anuncia que —en breve— el 50% del tráfico de la red, corresponderá a vídeos. Además, el número de páginas existentes ya supera al de los habitantes del planeta y la gran mayoría de ellas siguen basándose en el viejo estándar (que, a su vez, tiene por núcleo principal lo ya establecido a primeros de los 90). Y veinte años son muchos años en Computación. La renovación se hacía imprescindible.

El problema de las fechas de terminación

El problema principal a la hora de construir algo tan universal como HTML 5, es precisamente, ese: su universalidad. Tiene que haber consenso, tiene que servir para todos, tiene que ser lo más sencillo posible y aportar novedades significativas… en suma: se necesita una gran cantidad de tiempo para poner todas las fuerzas en funcionamiento, cubrir todos los aspectos, realizar los paquetes de pruebas suficientes que aporten una garantía a lo que se aprueba, etc… Y el resultado —a primera vista— parece desalentador: la fecha prevista para la terminación definitiva supera el año 2020 (aunque, recientemente, se ha dicho que esa fecha se rebajará, incluso hasta 2014).

¿Qué sentido tiene entonces hablar de algo tan lejano de su terminación? Según Cotton, el estándar podrá estar totalmente operativo a comienzos de 2013 y, aunque la versión final candidata, —la llamada **Candidate Recommendation**— no se termine

de forma oficial hasta mucho después, el "corpus" principal del estándar debe estar listo —e implementado en los navegadores— mucho antes.

De hecho, la mayoría de los navegadores modernos ya lo implementa en buena parte y cada nueva versión aporta mayor compatibilidad. Esa es una de las razones por la que la política de actualizaciones en los navegadores ha sufrido un acelerón tan notable en los últimos tiempos.

Por otra parte, existen librerías para resolver las posibles incompatibilidades en el caso de navegadores más antiguos, simulando ese comportamiento, o bien aportando algo que se ha dado en llamar *fallback*: una salida alternativa cuando las cosas no van exactamente como se pensaba, o no existe soporte de una característica por parte de un navegador dado.

nota

La recomendación actual por parte de los fabricantes, dado el soporte desigual del estándar es, que —cuando se quiera implementar una característica nueva— se compruebe el soporte de esa característica, y no la versión del navegador: una práctica muy extendida en la Web.

En las referencias finales de esta obra añadimos un bloque de enlaces a algunas de las más utilizadas en la actualidad. Algunas, como **Modernizr**, tienen por objetivo, precisamente, facilitar la detección del soporte de las nuevas opciones, facilitando la labor de los desarrolladores.

Los navegadores y el estándar: IE10 como objetivo

No hace mucho, **Dean Hachamovitch**, jefe del equipo de desarrollo de Internet Explorer hablaba de muy altos porcentajes de implementación en la nueva versión IE10, que verá la luz junto al nuevo Windows 8. De hecho, la versión IE9 ya soporta una buena parte de la especificación en su estado actual. Faltan cosas, y hay otras pendientes de definir, pero, entre tanto, ya podemos aproximarnos a la parte principal de lo que va a ser HTML 5, gracias a las implementaciones que los fabricantes de navegadores están haciendo ahora mismo. Y lo mismo puede decirse de Chrome, FireFox y Opera.

Con esta situación se plantea un curioso reto (que no se había dado hasta ahora): saber quién es el que mejor "hace los deberes". De momento, IE9 parece aplicado,



pero, su sucesor quiere ser el primero de la clase. En las demos de IE10 Consumer Preview realizadas en eventos como Tech-Ed 2012, IE10 le daba "sopas con honda" a sus competidores, en lo referente al motor de interpretación de JavaScript (**Chakra**), jugando con elementos visuales, si bien, le falta todavía madurez al tratarse de una versión beta. La versión para HTML 5, que ya está presente en la versión Release Preview de Windows 8, no solo es la más rápida, sino que incluye un nivel de soporte de HTML5 y CSS3 sin precedentes.

Porque, cuando hablamos de nivel de soporte, nos referimos a los 3 pilares que dan estructura al estándar, y el soporte de los nuevos modelos propuestos para las Hojas de Estilo, junto a las API de JavaScript se convierte en fundamental. Esta nueva versión de JavaScript, está siendo terminada de forma simultánea por la entidad ECMA[11] (grupo de trabajo TCI39), y a su motor de interpretación de código hay que añadir un buen número de API que permitirán gestionar todos los aspectos necesarios no solo en los sitios Web actuales, sino en las aplicaciones web, tal y como se conciben hoy día (Geo-localización, Web Sockets, Almacenamiento de sesión y local, etc.).

Para tener una idea más exacta de la situación, probaremos las características más importantes de HTML 5, CSS 3 y las nuevas API de JavaScript en todos los navegadores populares en sistemas operativos Windows indicando (al momento de escribirlo), su grado de soporte: IE9, IE10, Chrome, FireFox, Opera y Safari. Esto incluye el soporte de la versión "Metro Style" de IE10, que solo puede usarse desde Windows 8. Por tanto, algunas de las pruebas están realizadas desde Windows 7 Ultimate, y todas ellas, se han probado en Windows 8 Release Preview.

Nuevas reglas del juego

El nuevo "motor" de interpretación de JavaScript (o ECMAScript, si se quiere), llamado Chakra, ya ofrece un rendimiento excelente para IE9, y los demás fabricantes están haciendo lo propio. Ese parece ser el juego. Pero, por primera vez, hay un árbitro en él, que puede ser la W3C, que tiene la voluntad de establecer las reglas de ese juego. Si esas reglas se cumplen, el futuro parece brillante: programar una vez, y tener el mismo

[11] http://www.ecma-international.org/

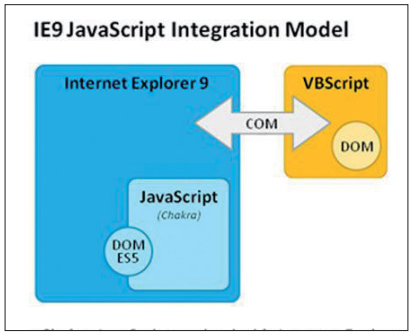


figura 1 Modelo de integración de scripts en IE9 basado en "Chakra"

resultado en todos los sitios (***one markup***). No importa el dispositivo de visualización, e incluso un cambio de plataforma. La página o aplicación debería verse/funcionar exactamente igual en cualquier parte.

En suma, la promesa universal de "programa una vez y consúmelo donde quieras", que —los veteranos como yo— hemos escuchado tantas veces en 25 años de historia práctica de la informática. Solo que esta vez, podría acercarse mucho a la realidad.

Se dan los medios para ello. Existe la tecnología, la voluntad de los fabricantes de seguir el estándar y la posibilidad de que los navegadores aprovechen al máximo el hardware de la máquina para una nueva experiencia de usuario mucho más rica en rendimiento y en posibilidades.

Los navegadores antiguos

¿Y los viejos navegadores? Esa es la cuestión. Los navegadores antiguos no soportan nada de esto y solo las últimas versiones dan cobertura de conjuntos más o menos completos de HTML 5. Por suerte, como hemos apuntado antes, hay una solución.

Conscientes de esta necesidad, las empresas y muchos desarrolladores independientes están generando cada vez más librerías que simulan mediante JavaScript el comportamiento del estándar para ellos. Al final de esta obra, hacemos una recopilación de los más atractivos y que más implantación están teniendo en la comunidad. En cualquier caso, será el programador Web el que tiene que decidir qué características le interesan persistir "hacia atrás", y qué necesita incluir para dar ese soporte de compatibilidad o *fallback*, que indicábamos antes.

Los objetivos del estándar en la práctica

Cotton era bastante claro a la hora de sintetizar los objetivos de estos esfuerzos de universalización: conseguir la Plataforma Web Abierta (*Open Web Platform*). En el camino, esto puede desglosarse en una serie de objetivos más sencillos y específicos. Voy a permitirme añadir una lista de algunos de los que los responsables parecen subrayar en sus intervenciones públicas:

- **HTML 5**. Un nuevo lenguaje de marcado que recoja las necesidades actuales: audio y video, nuevo sistema de gráficos y animaciones, etiquetas semánticas que colaboren con los robots SEO, unificación de criterios y eliminación de etiquetas redundantes, compatibilidad hacia atrás, y sobre todo el aspecto del consenso que ya hemos citado.

- **CSS 3**. Un nuevo lenguaje de definición de estilos, coherente con el de marcado, que coopere con el anterior a la hora de facilitar la separación del contenido de la presentación e, igualmente, recoja las necesidades actuales.

  Como objetivo secundario, está claramente el de ofrecer alternativas a soluciones de tipo gráfico y dinámico que solo podían ser resueltas mediante complementos (*add-ins* del tipo Flash o Silverlight). Por ejemplo, las aplicaciones Windows 8 hacen uso extensivo de estas capacidades.

- **ECMAScript-262 5ª Edición.** El nombre JavaScript es más un apodo que un nombre real (canónico, diríamos). Inicialmente, se llamó **LiveScript**, aunque, posteriormente, por una decisión estratégica entre Netscape y Sun MicroSystems pasó a llamarse así. Por otro lado, la entidad de estandarización de JavaScript es ECMA (*European Computers Manufacturers Association*)[12], y la especificación "oficial" de JavaScript no es otra cosa que el estándar ECMA-262, que vio su primera versión en 1997, aunque no fue hasta 1999, cuando se publicó la 3ª Edición, y comenzó su adopción de forma masiva. A partir de ahí, los que lo han implementado han construido diversas variantes del lenguaje, aunque la proliferación ha hecho una gran labor de unificación en los últimos años. Por si esto fuera poco, la 4ª edición se abandonó por diferencias políticas en la forma de imple-

[12] http://www.ecmascript.org/

mentación. Así que, buena parte del trabajo realizado para esa versión 4 (el que disfrutaba de un mayor consenso) ha pasado a formar parte de la nueva edición.

La idea de conseguir una nueva versión del lenguaje JavaScript que funcione paralelamente con HTML 5 y CSS 3, que incluya ciertas novedades, y mejore otros aspectos ya en uso, como la notación JSON, es uno de los objetivos de este esfuerzo colectivo.



El pasado año (junio/2011), ECMA anunciaba la aprobación de la 5ª edición de su estándar que está disponible para descarga como documento PDF[13]. 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.

Además, los test requeridos para su validación se realizan de forma independiente en otra entidad, la TC39 (o *Ecma Technical Committee 39*), que dispone de un sitio dedicado[14] en el que cualquiera puede colaborar si lo desea.

Motores de JavaScript

El motor de interpretación de JavaScript es algo delicado, porque —estando hecho independientemente por cada fabricante— debe respetar el estándar de la misma forma. Existen más de una docena de motores implementados en software diverso, de los cuales solo 7 incluyen compilación JIT (*Just-In-Time*), lo que les hace mucho más adecuados para la optimización del rendimiento y el aprovechamiento del hardware de la máquina: **Carakan** (Opera), **Chakra** (IE9/IE10), *V8* (Chrome), **SpiderMonkey** (Firefox), **SquirrelFish** —o Metro— (Apple WebKit), **JavaScriptCore** (Safari) y **Tamarin** (Adobe Flash).

- **Las API relacionadas.** HTML y sus tecnologías no aportarían mucho al objetivo de construir aplicaciones basadas en el estándar si no fuera por la presencia de API que aporten lo que ningún lenguaje de marcas o definiciones

[13] http://www.ecma-international.org/publications/standards/Ecma-262.htm

[14] http://test262.ecmascript.org/

puede aportar. Y esto se resume, sobre todo, en la gestión del estado de las aplicaciones, la geo-localización, las aplicaciones *off-line*, las comunicaciones dúplex, el almacenamiento local y de sesión, el acceso a una base de datos sencilla, el soporte de tareas asincrónicas, etc.

Como hemos apuntado, dichas tareas no están siendo abordadas por la W3C directamente. Ellos, más bien, sirven de coordinadores del trabajo, como explica **Paul Cotton** en su entrevista. La buena noticia es que se han establecido ya claramente cuáles van a ser las más importantes, y las que formarán parte de la versión final, y cuales no (por ejemplo, **WebSQL** se ha abandonado a favor de **IndexDB**).

nota

Conviene distinguir las ´ PI del lenguaje en sí, aunque la forma de programarlas sea mediante JavaScript. Las ´ PI 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.

Por ejemplo, el atributo `type` de la etiqueta `<input>` ahora dispone de un conjunto nuevo de valores posibles, como: `email`, `tel`, `date`, `time`, etc. Podemos usar un API asociada programable en JavaScript para validar dichas entradas, o podemos simplemente programar su atributo `required` para que, si el usuario no introduce un dato en este campo de formulario, reciba una notificación personalizada.

De igual forma, hay escenarios cuya programación requiere esencialmente de código JavaScript, como sucede en procesos de comunicación donde existe suscripción a eventos o comunicación dúplex con servidores.

Librerías y lenguajes que compilan a JavaScript

Ante la perspectiva de esta nueva Web de sitios y aplicaciones, se está dando un renacimiento del lenguaje JavaScript que, con todas sus carencias, es uno de los lenguajes más utilizados del mundo, precisamente por su presencia en la red. Esas carencias están provocando la ascensión y adopción popular de librerías complementarias, como **jQuery**[15], que facilitan muchísimo el uso de JavaScript, así como de otras que comple-

mentan carencias de navegadores antiguos, o facilitan la inclusión de elementos de interfaz de usuario más actuales (como Kendo-UI, de Telerik), y muchas más.

Pero uno de los movimientos más interesantes, está sucediendo en el apartado de los llamados "Lenguajes que compilan a JavaScript". Se trata de herramientas y/o lenguajes que incorporan notables mejoras en lo operativo y lo sintáctico, y que finalmente, generan código JavaScript tal y como sucede con la propuesta de Google respecto a su lenguaje propietario **Dart**, del que ya existe un sitio Web oficial[16]). Existen muchos lenguajes para los que se dispone de esta posibilidad: Objective-C, Ruby, Python, Perl, Java, Scala, C#, Lisp, oCaml, Haskell, SmallTalk, C/C++, Basic, Pascal, Multitarget, CoffeeScript y otros. De todos ellos podemos encontrar versiones actualizadas, por lo que si el lector piensa abordar una aplicación con abundancia de código final en JavaScript, un vistazo a los posibles candidatos no estaría de más[17].

En concreto, dentro del marco de .NET, disponemos de librerías como **JSC**, **JSIL**, **Prefix** o **Script#**. Salvo esta última, las 3 primeras son gratuitas. Muchas de estas herramientas permiten plantear problemas en nuestro lenguaje favorito .NET (suelen soportar VB.NET, C# y F#), y generar posteriormente un código fuente en JavaScript, que puede funcionar en cualquier navegador (al menos, a priori).

Otros funcionan sobre ensamblados, siendo capaces de crear librerías JavaScript a partir de librerías (DLL) en .NET. En el capítulo 2 incluiremos referencias a éstas y otras posibilidades.

¿Por qué no esperar a que el estándar esté terminado?

Terminado o no, los fabricantes están incorporando el soporte, dado que la posibilidad de consumir servicios de la nube mediante una sola programación es muy atractivo y promete abundantes ingresos. Y esto es válido con un énfasis especial en los 5 navegadores más populares: IE, Firefox, Google Chrome, Safari y Opera.

Así que se trata también de una decisión estratégica. Podemos migrar nuestros sitios a HTML 5 ahora mismo, y empezar a aprovechar sus posibilidades. Las librerías de compatibilidad, y las otras opciones apuntadas, consiguen que los navegadores antiguos comprendan el modelo (casi) de igual forma, y de esa manera, estaremos pre-

[15] http://jQuery.com
[16] www.dartlang.org
[17] Puede encontrar una completa lista de las opciones disponibles y sus enlaces asociados en la página http://bit.ly/goZv7i

nota

Como comentaremos más adelante, la apuesta por el estándar de algunos fabricantes, como Microsoft, es tan fuerte, que Windows 8 puede programarse directamente mediante éste nuevo estándar. Hablaremos más de ello en éste capítulo.

parados para las evoluciones futuras y podremos aprovechar sus características globales. Y hay otras ventajas adicionales.

De cara a los buscadores, la precisión con la que éstos pueden encontrar nuestras páginas debido a las características semánticas, será muy superior. Podremos utilizar jQuery para enriquecer las páginas, obteniendo un buen rendimiento gracias a los nuevos motores y no precisaremos de ningún añadido adicional si lo que queremos es, simplemente, mostrar un vídeo o reproducir una animación, por ejemplo.

Y contamos con otra consideración a favor de la adopción inmediata. Las herramientas de desarrollo que propondremos a lo largo de esta obra (Visual Studio 2012 - o 2010 y las versiones Express), ya soportan la sintaxis de los estándares y podremos encontrar un valor añadido en ellas (el nuevo `Page Inspector` de la versión 2012 es excelente para el desarrollo con el estándar).

El sueño de una Web semántica



De esta forma, una de las promesas inherentes al nuevo estándar es la de dar un paso más hacia el sueño de una Web Semántica. Una Web con inteligencia, basada en el significado y el contexto. El término se acuñó en 2001 como resultado de un artículo escrito por **Tim Berners-Lee**, que describía el término como *El lugar en el cual las máquinas pueden leer páginas Web con la misma facilidad con la que los humanos lo hacemos*. Algunos la denominan la Web 3.0.

Para poder profundizar algo más en estos términos, conviene antes recordar o aclarar algún aspecto vinculado con ella.

Los Identificadores uniformes de recursos (URI vs. URL)

Si quiero hablar de algo, en primer lugar debo identificarlo. Puede hacerse de manera indirecta: "La boca del metro", "El gato del garaje", o "Las cosas que le gustan a Pepe". También podría elegir a ser más directo: "W3C", "Scott Guthrie" o "Megajuegos".

Para identificar elementos en la Web, también utilizamos identificadores. Se usa un sistema uniforme de identificadores, y cada elemento identificado es considerado un "recurso", así que llamamos a estos identificadores *Identificadores uniformes de recursos* o URI para abreviar. Podemos asignar un URI a cualquier cosa, y cualquier cosa que tenga un identificador URI puede decirse que está "en la Web": usted, el libro que compró esta mañana, el último electrodoméstico, etc., todos ellos pueden tener un URI. El identificador URI es el fundamento de la Web. Aunque se podría reemplazar casi cualquier otra parte de la Web, no se podría prescindir de los URI: mantienen unida toda la Web.

El lector ya está familiarizado con una de sus formas: la dirección URL o localizador uniforme de recursos. Una URL es una dirección que le permite visitar una página Web, tal como: `http://www.w3.org/standards/webarch`.

Al comienzo, vemos que una dirección URL le dice al navegador donde encontrar un recurso específico (en este caso, el sitio Web de dirección del W3C, pero también puede ser un gráfico, un archivo PDF o cualquier otro). A diferencia de otras formas de URI, una URL identifica y localiza. Esto lo diferencia de la URI. La URI identifica un mensaje de correo electrónico, pero no es capaz de localizar una copia del mensaje.

Como la Web es demasiado grande para controlarla, los URI están descentralizados. Nadie en concreto controla quién los hace o cómo se pueden utilizar. Mientras algunos esquemas URI (como `http:`) dependen de sistemas centralizados (como DNS), otros sistemas (como `freenet:`) están totalmente descentralizados.

URI y semántica

Ahora bien, como cualquier persona puede crear un identificador URI, inevitablemente acabaremos con varios identificadores URI que representan lo mismo. O lo que es peor: no habrá ninguna forma de averiguar si dos identificadores URI apuntan exactamente al mismo recurso. Por lo tanto, nunca podremos decir con certeza lo que significa un URI concreto. Se precisan soluciones de compromiso para crear algo tan enorme como la Web semántica.



Un URI no es un conjunto de direcciones diciendo a su equipo como llegar a un archivo específico en la Web (aunque también puede hacerlo). Es un nombre de un "recurso" (algo). Este recurso puede o no ser accesible por Internet. Incluso podría darse el caso de que la URL tuviese un nombre totalmente "oclusivo" pensado precisamente para ocultar su auténtico contenido en vez de sugerirlo.

Así pues, un primer paso (que ya se está empezando a dar) es hacer que las propias URL expliquen de la forma más clara posible cuáles son las palabras clave de su contenido, diseñando nombres de página compuestos de lo que podríamos llamar palabras clave de la página. Es un primer paso. Los siguientes, supuestamente, dependerán de un buen uso de las nuevas etiquetas de HTML 5 en relación con el contenido que albergan.

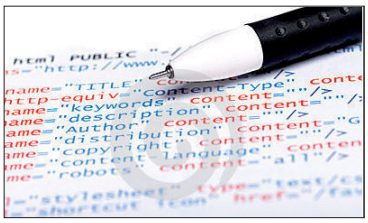
Por su parte, Wikipedia define la Web Semántica con las siguientes palabras: *Se basa en la idea de añadir metadatos semánticos y ontológicos a la World Wide Web. Esas informaciones adicionales —que describen el contenido, el significado y la relación de los datos— se deben proporcionar de manera formal, para que así sea posible evaluarlas automáticamente por máquinas de procesamiento. El objetivo es mejorar Internet ampliando la interoperabilidad entre los sistemas informáticos usando "agentes inteligentes". Agentes inteligentes son programas en las computadoras que buscan información sin operadores humanos.*



Nótese que he subrayado la palabra "añadir" entendiendo que —para aquellos héroes que no tengan inconveniente en rescribir su código— podríamos convertir con cierta facilidad un sitio existente en un sitio con capacidades semánticas con solo añadir unas etiquetas a la página, que, por otra parte no son muchas, pero, como veremos a partir del Capítulo 3, ayudan a estructurar los documentos de forma que los robots buscadores "comprendan" las relaciones en las diferentes partes de un sitio o un página completa y ofrezcan resultados más apropiados. Son esos *Agentes inteligentes* que se mencionan en la definición.

Las propuestas iniciales -basadas en el estándar RDF— no tuvieron demasiada acogida, y, una vez más, cabe la posibilidad de que una política de "hechos consumados" llegue a buen fin, o al menos, a dar un paso importante en esa dirección. Para ello, como veremos en el apartado de HTML, el nuevo estándar propone una decena de etiquetas que son consideradas mejoras en las capacidades de descripción del contenido, y por tanto, facilitarán los resultados coherentes en las búsquedas en un futuro cercano.

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. 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, es mediante la colaboración y las aportaciones de sus miembros directos, que se encargan de sugerir ideas y también de revisar las contribuciones y sugerencias de los participantes de carácter abierto, que son varios cientos en todo el mundo[18]. 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, hay más de 100 especificaciones agrupadas bajo el concepto (o la marca) HTML5 y podrían incrementarse en el futuro antes de llegar al estado Last Call. Pensemos 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 relacionadas y ya aprobadas o en fase de serlo, etc. Esto plantea un serio reto respecto a las pruebas de funcionamiento.

Los test

Una vez identificados y aprobados inicialmente, con los materiales reunidos se procede a testar el nuevo estándar en varias fases de pruebas. Internamente, la W3C dispone de una herramienta denominada **Bugzilla**, con la que gestionan el estado de los bugs y su situación respecto a la resolución.

[18] 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**.

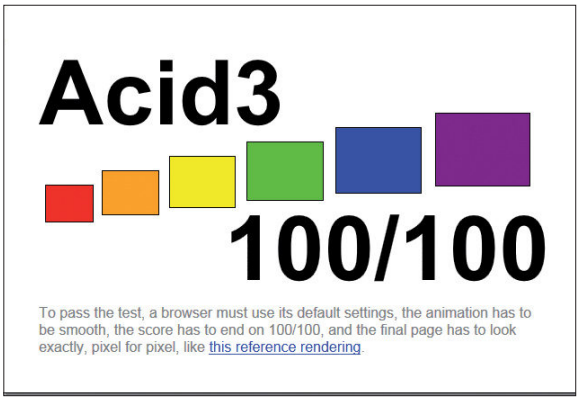


figura 2 Todos los navegadores utilizados en las pruebas (IE9/10, Chrome 14, Firefox 7, Safari 5.1 y Opera 11.5 pasan los test ACID al 100%.

A partir de aquí, la W3C y sus entidades asociadas comienzan los test oficiales, para los que utilizan una serie de herramientas propias, a la vez que coordinan iniciativas individuales, "extraoficiales", con el propósito de elaborar cuadros de conformidad respecto a las normas establecidas.

Los sitios alternativos de pruebas de conformidad

La validez de los sitios de pruebas que están fuera de los canales oficiales es más que discutible, y algunas veces roza lo partidista o, simplemente, interesado. Por ejemplo, algunas prueban aspectos que no están en el estándar, o dan la misma validez a aspectos colaterales o parciales que a otros troncales de la especificación. De este corte, podemos catalogar algunas propuestas como `HTML5Test.com`, que testea otros aspectos que no son parte del estándar, o `Caniuse.com`, que no comprueba la implementación correcta de las características sino solo su existencia.

Precisamente, teniendo en cuenta estas circunstancias, los creadores de uno de los test privados más populares (los test ACID) han modificado recientemente algunos aspectos de estos test, para que resulten más acorde con las especificaciones oficiales. Tras esas modificaciones, es interesante notar que todos los navegadores utilizados en esta obra pasan estos test con un 100% de efectividad.

Los sitios oficiales

De cualquier forma, los test más serios de compatibilidad con el estándar son los que provienen de sus creadores. En concreto, la página *HTML5 Test Suite Conformance Results*[19], realiza un test de compatibilidad para cada visitante indicándole el nivel de conformidad del navegador elegido respecto a 7 grupos principales, y mostrando después un desglose de más de 900 test, uno a uno. El lector puede probar este punto en la dirección indicada.

Además, para aquellos interesados o curiosos respecto a determinado aspecto del estándar, existe una página de la propia W3C que permite realizar test individuales de una sola característica (*test harness*), disponible en la dirección **`http://w3c-test.org/html/tests/harness/harness.htm`**.

Por otra parte, desde la aparición de la versión Release Candidate de Windows 8, ya disponemos de una versión del navegador IE10 muy próxima a la que será la final, y precisamente, Microsoft ha publicado casi simultáneamente un análisis de soporte del estándar en comparativa con las versiones más recientes de los navegadores, estableciendo como fecha el 1 de junio de 2012. El gráfico siguiente muestra los resultados del porcentaje de soporte de ésta versión, en comparación con las versiones más recientes (a la misma fecha) del resto de navegadores populares: Chrome, FireFox, Opera, Safari e IE9. Las diferencias pueden apreciarse a primera vista, ya que, del total de test enviados, —en 7 categorías distintas— el nuevo IE10 obtiene el 100% en 4 de ellas, el 99% en otras 2, y el 96% en la restante.

Como puede apreciarse en la figura 3, tomada del sitio oficial de test para IE[20], en estos momentos IE10 es el navegador que implementa el estándar HTML con mayor nivel de fidelidad.

Web Standards	Number of Submitted Tests	Internet Explorer 10 Release Preview	Mozilla Firefox 12	Opera 11.64	Apple Safari 5.1.7	Google Chrome 19.0.1084.46	Internet Explorer 9
10 New! HTML5	326	99%	74%	64%	74%	89%	36%
SVG 1.1 2nd edition	74	100%	93%	93%	88%	97%	95%
20 New! CSS	236	100%	70%	59%	63%	66%	62%
DOM	127	100%	89%	85%	90%	94%	100%
Web Performance	28	96%	89%	0%	0%	89%	39%
37 New! Web Applications	464	99%	78%	42%	32%	54%	21%
8 New! JavaScript	6,318	100%	97%	99%	91%	100%	91%

figura 3 Resumen de la evaluación de test del sitio dedicado de Microsoft.

[19] http://w3c-test.org/html/tests/reporting/report.htm
[20] http://samples.msdn.microsoft.com/ietestcenter/

En esa dirección, además, se encuentra permanentemente actualizado un documento indicando el estado actual de los test que se renueva con mucha frecuencia y establece el número de test enviados para control, así como el número de ellos que los grupos de trabajo del comité ha conseguido evaluar y han pasado las pruebas. En concreto, el *HTML5 Testing Task Force Status*, ofrece un resumen informativo como el de la figura 4.

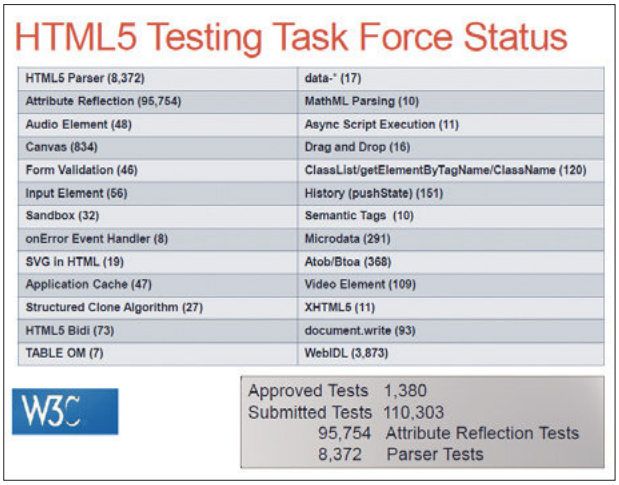


figura 4 Documento de seguimiento del estado de los test oficialmente pasados con un resumen de los ítems que se consideran aprobados y el total de pruebas enviadas.

Y, en general, para cualquier aspecto que el desarrollador necesite comprobar sobre una característica o API del estándar, puede acudir a la página oficial de pruebas (`http://w3c-test.org`) donde encontrará toda la información adicional.

Las aplicaciones web y el nuevo modelo de aplicaciones en Windows8

En el evento BUILD/2011, Microsoft presentó su nuevo modelo de aplicaciones vinculado con el sistema operativo Windows 8. Más recientemente, en los eventos Tech-Ed de EE.UU. y Europa[21], se ha ofrecido información más madura y concreta sobre la pro-

[21] http://europe.msteched.com/



gramación para esta nueva plataforma. Las implicaciones son de amplio calado, y no vamos a profundizar aquí en el tema, pero si me gustaría recalcar algunos aspectos que afectan directamente a la utilización del estándar en este modelo de aplicaciones.

Como el lector probablemente conocerá, la nueva interfaz de usuario de Windows 8 es lo más llamativo a primera vista, ya que propone cambiar el paradigma habitual que se ha estado manteniendo desde las primeras versiones de Windows por uno nuevo, especialmente preparado para pantallas táctiles y se inspira totalmente en la interfaz de usuario que presentó el sistema operativo para móviles Windows Phone 7.

Aplicaciones Windows 8

Para este tipo de aplicaciones se requerirá la programación mediante un nuevo modelo de aplicaciones, en el que la parte de interacción manual con la pantalla del usuario es fundamental, y se debe tener muy en cuenta en el desarrollo desde los inicios del ciclo de vida de la aplicación.

En resumen, el nuevo sistema (que aparecerá a finales de octubre de 2012), propone



figura 5 Aspecto inicial de Windows 8 en ejecución mostrando la nueva interfaz.

dos tipos de aplicaciones: las tradicionales, que seguirán funcionando como hasta ahora y las Windows 8 (ver figura 5), en las que se hace hincapié en una nueva forma de utilización donde las pantallas son de tipo táctil, y una API llamada WinRT (*Windows Run-Time*) provee a las aplicaciones de lo necesario para acceder a los recursos del sistema.

Se trata de una capa muy ligera que funciona por encima del *kernel* de Windows, y tiene la particularidad de permitir a desarrolladores de diversos lenguajes un acceso común y uniforme a esos recursos, independientemente de que provengan del mundo .NET, de C/C++, o de HTML5 y JavaScript.

El nuevo modelo de aplicaciones Windows 8

Por tanto, se nos plantea un modelo de construcción de aplicaciones nuevo (con un nuevo estándar de distribución a través de un *MarketPlace* o tienda *on-line*) que propone un esquema conceptual como el que vemos en la figura 6.

Como se ve en el diagrama, la capa **WinRT** alberga varios grupos de librerías, organizados

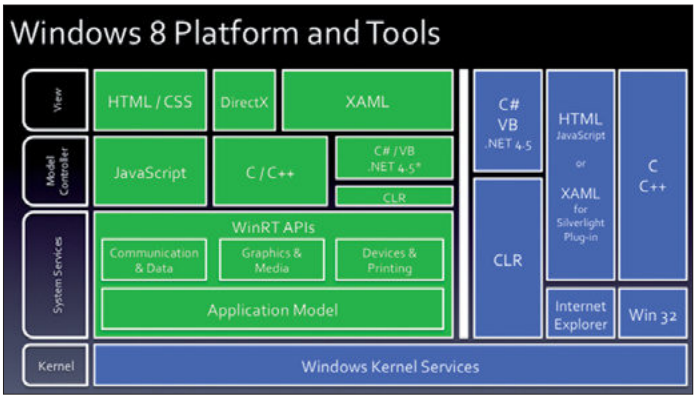


figura 6 Plataformas de desarrollo y herramientas para Windows 8.

por criterios de funcionamiento (Datos y comunicaciones, Gráficos y Multimedia y Dispositivos e Impresoras). Todas las aplicacione Windows 8, se construyen mediante el nuevo modelo de aplicaciones (*Application Model*) que accede a la capa ***WinRT*** para obtener

recursos del sistema operativo. Es lo que vemos en el gráfico en la parte inferior de la zona verde, y que se agrupa como System Services (a la izquierda, en vertical).

Por encima tenemos los controladores de modelo (*Model Controllers*), o sea, los motores de ejecución y las API en las que nos apoyaremos para crear este tipo de aplicaciones, dependiendo de la opción de lenguaje escogida.

Finalmente, en la capa superior, de esta zona verde (o zona Windows 8), bajo el epígrafe "View", es donde aparecen los diversos lenguajes, y vemos 3 opciones disponibles: HTML y CSS, XAML/DirectX con C/C++ y XAML con .NET.

La posición de Silverlight y Flash en Windows 8

Advierta el lector que Silverlight no aparece en esta área, y —de hecho— está expresamente vetado ya que no se podrán ejecutar complementos desde el navegador (ni Silverlight, ni Flash), ni se podrán ejecutar aplicaciones OOB en este contexto.

¿Significa eso que nuestras inversiones anteriores no serán válidas? En absoluto. Como apreciamos en el gráfico, la zona azul se reserva para la ejecución de todas las aplicaciones anteriores, y ahí tiene cabida Silverlight, igual que cualquier otra aplicación realizada en .NET, C/C++, o incluso, HTML5 y JavaScript, que —como vemos— se encuentra en ambos lados del diagrama.

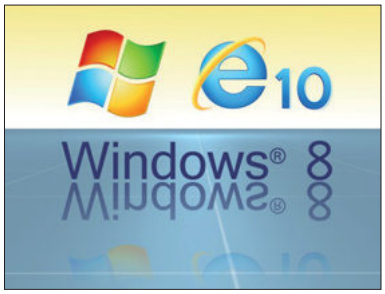
Adviértase también que —no estando Silverlight ni WPF específicamente indicados en la zona verde— XAML se convierte en un modelo común para la construcción de interfaces de usuario, tanto para aplicaciones basadas en .NET como para los desarrolladores de C/C++, si bien éstos pueden usar igualmente DirectX.

O sea, que nuestra inversión previa en aprendizaje, código generado, librerías corporativas, etc., se encuentra garantizada por esta arquitectura con mínimos cambios. En caso de preferir continuar con las aplicaciones tradicionales de escritorio, o simplemente, instalar o mantener aplicaciones tradicionales, nos moveremos en la "zona azul", donde todo lo presente hasta ahora sigue apareciendo, y para lo que el sistema garantiza un funcionamiento transparente.

WinJS y los controles "de fábrica"

Respecto a esta diferencia en los tipos de aplicaciones, la división parece nacer de una separación conceptual que no se había dado mucho hasta ahora, y que consiste en separar totalmente las aplicaciones de gestión, de las "aplicaciones Internet". Éste último modo se recomienda especialmente para el modelo de aplicaciones web (que no sitios

web), que emularán la funcionalidad de las clásicas, pero ejecutándose desde el navegador, y apoyándose en las nuevas API del estándar HTML 5.



Ahora bien, para que este modelo de aplicaciones web funcione, se requiere un mecanismo que comunique el código JavaScript, interpretado por el motor Chakra, con la capa WinRT y ahí es donde aparece un nuevo protagonista: **WinJS**. WinJS (o Windows desde JavaScript) es la librería para construir aplicaciones tipo Windows 8 con JavaScript.

Esta librería, facilita el diseño de estas aplicaciones y dispone de controles para las llamadas "experiencias comunes de desarrollo", estando diseñado, tanto para aplicaciones táctiles como para introducción de datos tradicional, además de ser muy escalable. Contiene un enorme conjunto funcional pensado a tal efecto: animaciones, plantillas, utilidades, estilos CSS predeterminados, controles, mecanismos de *binding*, modelos de aplicación, gestión de la navegación, fragmentos, *promises* (para procesos asíncronos), y un largo etcétera. Como muestra de los controles predeterminados véase la imagen siguiente (figura 7).

De forma que WinJS es la librería que tiende el puente con **WinRT** para llamar

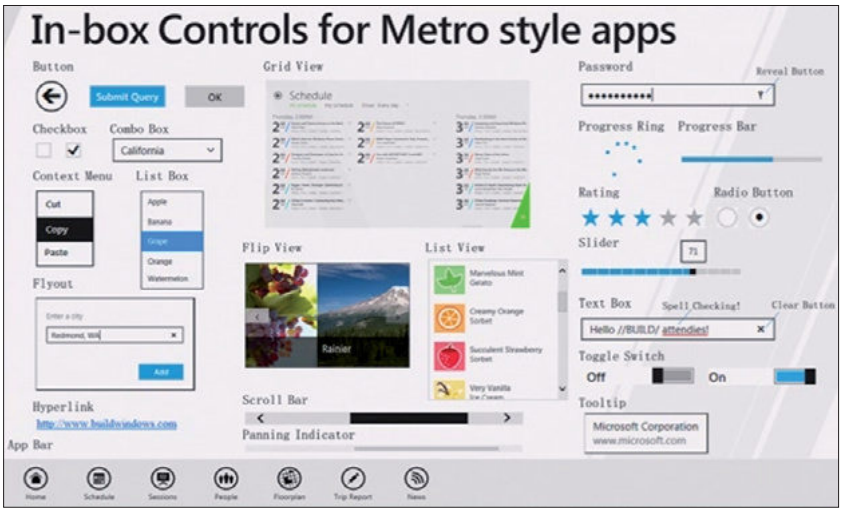


figura 7 Los controles para uso con JavaScript disponibles en el IDE de Visual Studio/Blend

a las API necesarias del sistema. De hecho, cuando desarrollamos una aplicación Windows 8 con JavaScript, así se llama el espacio de nombres que nos da acceso a WinRT.

Si el lector descarga la versión Windows 8 Consumer Preview, e instala la versión gratuita por tiempo limitado de Visual Studio 2012, podrá iniciarse en la construcción de aplicaciones de esta clase. Un vez instalado, en la opción "Nuevo Proyecto", veremos cómo Visual Studio nos ofrece un paquete de opciones de diversas plantillas preinstaladas —a su vez— separadas por categorías, como podemos ver en la figura 8.

Si optamos por una de ellas, por ejemplo la de Geo-localización, veremos que nos

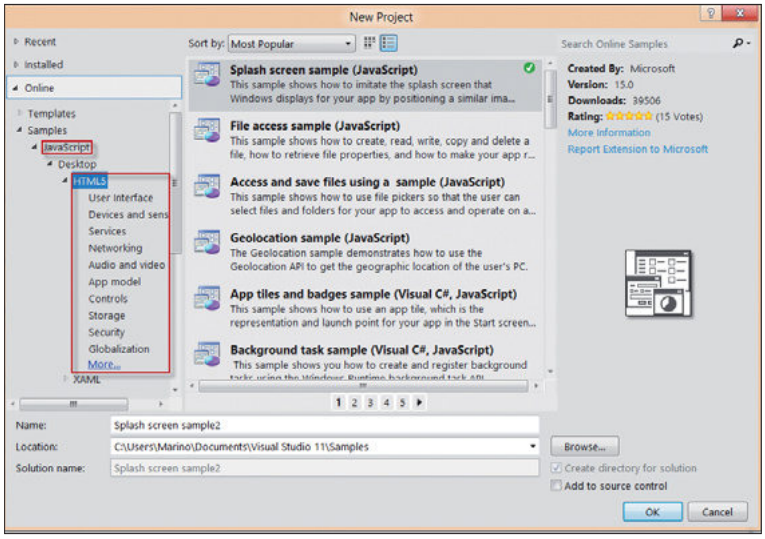


figura 8 Tipos de plantillas de proyecto asociadas a JavaScript dentro de Visual Studio 2012

construye una aplicación básica de tipo Windows 8, que incluye las características de esta API. Y, si accedemos al archivo que contiene el punto de entrada de la aplicación (**`program.js`**), veremos código similar al del listado 1.

Donde advertimos la llamada de inicialización de la aplicación, de forma similar a lo que los desarrolladores de .NET estamos acostumbrados en las llamadas a **`InitializeComponent()`** de las clásicas aplicaciones .NET. Se trata ya de aplicaciones programadas bajo el nuevo estándar, que hacen uso de los 3 pilares principales: HTML5, CSS3 y JavaScript.

Si observamos el Explorador de soluciones, veremos que, en "References", aparece la librería Microsoft Windows Library for JavaScript SDK, que es la que se utiliza para

```
function initialize() {
  WinJS.Application.start();
  WinJS.Application.addEventListener("checkpoint", applicationStateCheckpoint, false);
  WinJS.Application.addEventListener("activated", applicationActivated, false);

// Make the details container in the input panel selectable so
// that end-developers can copy and paste the text
        var detailsContainer = document.querySelector("#input .details");
        if (detailsContainer) {
            detailsContainer.setAttribute("data-win-selectable", "true");
        }
```

Listado 1: Código JavaScript generado por Visual Studio 2012 que utiliza WinJS

el acceso a los recursos del sistema.

En el ejemplo que hemos utilizado, este paquete contiene realmente otros 4 archivos JavaScript, especializados en distintas funcionalidades: **base.js**, **base.strings.js**, **ui.strings.js** y **ui.js**: algunas sirven de fundamentos para la construcción y acceso a los recursos del sistema, y otras son específicas de ciertas operaciones, como las relacionadas con la IU.

El resto de archivos de la aplicación contiene la pantalla principal (**default.html**), la versión WinRT de los archivos **.manifest** de configuración y unas hojas de estilo para manejar los aspectos de presentación en pantalla. La figura 9 presenta la vista del Explorador de soluciones para una aplicación de esta clase sin modificar.

La división granular de JavaScript

El usuario acostumbrado a otros tipos de aplicaciones seb, como MVC4 o Web Forms, quizá se sorprenda de la cantidad de "pequeños ficheros" JavaScript que la aplicación ubica, de inicio, en distintos directorios de la solución.

Esto tiene que ver con una nueva característica presente en MVC 4: *bundling and minifying*, que permite compactar todos los archivos JavaScript en uno solo, eliminando los espacios y reduciendo el número de peticiones al servidor, así como el tiempo de carga.

La razón de tanta granularidad es la nueva opción de JavaScript 5 llamada **Strict Mode**. Con el uso de esta modalidad, JavaScript 5 aporta interesantes novedades y modos de programación más acordes con las buenas prácticas presentes en otros lenguajes.

Pero hay un problema. Si un solo fichero de esta clase declara el uso de esta modalidad fuera de una función, al producirse el empaquetado, todos los demás van a asumir la interpretación estricta de forma automática, lo que podría desembocar en comportamientos no deseados. Mediante la granularidad se minimiza este efecto.

Ejecución y soporte del estándar

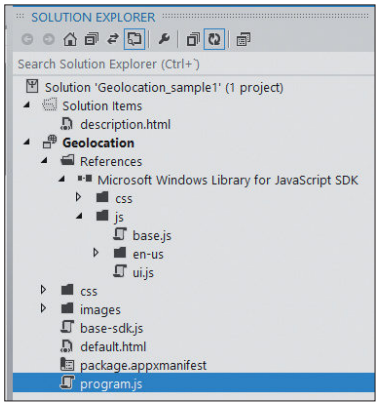


figura 9 Librería JavaScript del SDK para Windows 8 desde Visual Studio 2012

La aplicación compila y se ejecuta correctamente. Se trata de aplicaciones sin ventana (pantalla completa), que podemos manejar también con el ratón, a parte de las posibilidades de uso de pantalla táctil que hemos apuntado antes. La salida de esta aplicación una vez otorgado el permiso para ver los datos de geo-localización, es la que se corresponde con la figura 10:

Como se ve en la zona azul del esquema de los modelos de aplicación que hemos apuntado antes, también se dispondrá de un triple modelo para construir aplicaciones



figura 10 Salida de la aplicación estándar de geolozalización en Windows 8 Consumer Preview.

tradicionales o de escritorio en Windows 8. Si bien, en este caso, no se precisa WinRT y será el motor propio de IE10 el encargado de gestionar su ejecución.

Como era de esperar, el soporte de JavaScript en Visual Studio 2012 ha sido mejorado notablemente, para hacer que la experiencia de desarrollo sea lo más similar a lo que los desarrolladores de .NET estamos acostumbrados. Hablaremos con más detalle de esto, en el siguiente capítulo, dedicado a las herramientas de desarrollo y depuración con el estándar.

Los motores *Chakra* y los dos contextos de ejecución

A la vista de los dos tipos de aplicaciones y las dos formas de ejecución, parecen surgir ciertas discrepancias o imprecisiones respecto a los motores de HTML5/JS que aparecen repetidos en ambos lados del diagrama. En realidad, no es así. Simplemente, si queremos que una aplicación HTML5/JS se ejecute como una aplicación Windows 8, utilizaremos las librerías de WinJS. Si no queremos depender de Windows 8, y deseamos una ejecución universal en cualquier tipo de plataforma y dispositivo, utilizaremos el motor del navegador correspondiente.

Y el encargado de esa interpretación en I.Explorer es una versión renovada del motor Chakra actualmente en uso en IE9. Chakra (cuyo modelo funcional vemos en la imagen adjunta), es el nombre del motor de JavaScript/DOM altamente optimizado que Microsoft introdujo en su navegador Internet Explorer 9 y que ha seguido evolucionando desde entonces.

Está presente ahí y lo estará independientemente en la versión IE10 de Windows 8 (en sus dos versiones, Windows 8 y Desktop). Los test realizados sobre Chakra lo

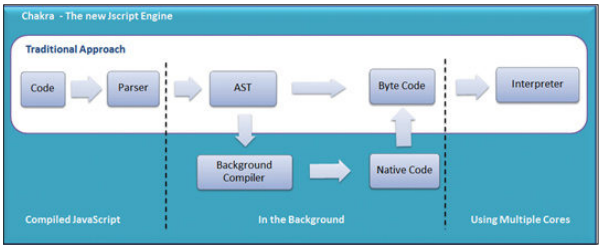


figura 11 Esquema operativo del motor de JavaScript Chakra para IE9/10

ponen a la cabeza de todos los motores existentes, ofreciendo un rendimiento sin precedentes en este tipo de aplicaciones, y un altísimo nivel de soporte de las API de JavaScript 5.

Así pues, este es el panorama que se nos presenta respecto al desarrollo con HTML 5. Muchas de las demos realizadas en los últimos eventos de desarrolladores (BUILD, Tech-Ed, etc.) estaban hechas utilizando estos recursos y ofrecían un rendimiento excelente en todas las máquinas utilizadas. Por otro lado, se comentaba que los cambios que habría que realizar para que una aplicación HTML5 funcionase con el nuevo modelo serían mínimos en un principio, y una aplicación hecha así, podría siempre extenderse al nuevo sistema con poco esfuerzo.

Al lector interesado, le recomendamos una visita a los sitios Web de los eventos de desarrollo de Microsoft indicados antes, si quiere acceder a vídeos explicativos de la forma en que se van a construir estas nuevas aplicaciones.

Hablando sobre el estándar: la opinión de un protagonista

Para concluir este capítulo y dar al lector una idea práctica de lo que algunos responsables vinculados con la construcción y la promoción del estándar piensan, nada mejor que sus opiniones en vivo. Presentamos una entrevista que realizamos a uno de los responsables directos de la creación de HTML 5: **Paul Cotton**, a quien, aprovechando su paso por Madrid, pudimos interrogar acerca de su trabajo dentro del World Wide Web Consortium (W3C). Paul ha trabajado en IBM, en las Naciones Unidas (en calidad de embajador) y, en la actualidad, como experto en lenguajes de marcas, representa a los grupos de trabajo de Microsoft que están colaborando en la confección del estándar.

Entrevista con Paul Cotton

A continuación transcribimos el contenido completo de la entrevista concedida a dNM.

> *Estamos en el Hotel Ritz de Madrid, con Paul Cotton. Es una de las personas al cargo del W3C HTML5 Working Group en relación con Microsoft. Nos gustaría comenzar esta entrevista con una primera pregunta en relación con la participación de Jean Paoli en el estándar XML. Si no recuerdo mal, esa fue la primera aproximación de Microsoft al trabajo en la*

elaboración de los estándares de la Web. ¿Qué puedes comentarnos al respecto?

Sí. De hecho, yo trabajaba con IBM cuando empezaron los trabajos con XML. Trabajaba con bases de datos, y añadir capacidades de notación de datos estructurados se ha demostrado que ha sido un área muy importante. Así que Jean (y **Tim Bray**, al que citabas antes de la entrevista, y al que conozco personalmente, porque los dos somos canadienses y hemos trabajado en el área de HTML) y yo en esa época fue cuando empezamos a colaborar con la W3C. Luego yo continué para dirigir el grupo de trabajo de XML Query.



Y pienso que esa es una característica importante de W3C. Reúne a muy diversas comunidades: la industria, la academia, etc. Hoy tenemos las mismas circunstancias que entonces. Todas las áreas de la industria están implicadas, pero también hemos invitado a expertos independientes, desarrolladores Web, expertos en Accesibilidad, etc.

Así que pienso que lo que dices es justamente la situación: reunir esas distintas comunidades juntas es un aspecto muy importante para ganar interoperabilidad. Y esos primeros contactos sembraron lo que HTML5 puede ser al día de hoy.

¿Qué sucedió con XHTML 2.0? ¿Era demasiado ambicioso o demasiado inaceptable, en términos de compatibilidad hacia atrás?

¿Sabes? Cuando hablo sobre interoperabilidad y estándares, soy un tanto darwiniano. Y cuando hablamos de especificaciones técnicas, a veces pienso que sucede como en la evolución. Tenemos muy buenas ideas, pero unas sobreviven y otras no. Existen como dos grandes corrientes en W3C: los interesados en los navegadores y los del grupo de XML y derivados. Y, de alguna forma, se disgregaron en direcciones distintas. Algunos pensaron que debido al éxito de XML, ese era el camino a seguir con la Web, igualmente.

Y creo que es justo decir que W3C, quizás, cometió un error, concentrando tantos recursos en torno a XHTML, y no apareció una nueva versión de HTML durante bastante tiempo.

Pero, hace unos 4 años, que W3C decidió que los esfuerzos debían dirigirse hacia este nuevo estándar HTML5, algunos de los que habían abandonado la W3C regresaron y ahora, tenemos de nuevo la comunidad completa reunida, porque lo que apuntabas antes es precisamente la fuerza de W3C: reunir a la comunidad en

torno a un esfuerzo común. Una vez que se consiguió esto, el proyecto entero floreció de nuevo.

Los recursos de la W3C son limitados y decidieron concentrarse en lo que se llamó más tarde Plataforma Web Abierta (*Open Web Platform*), que incluye HTML 5 y otras especificaciones. De hecho, en la actualidad se pueden publicar webs utilizando la compatibilidad de ambos formatos. Nosotros lo llamamos el "formato políglota".

En un interview previo, con Philippe Le Hégaret, mencionaba usted que, en ocasiones, el público en general se refiere a HTML 5 e incluye especificaciones que no van a ser estandarizadas. ¿Es esa la situación actual?

Es una pregunta muy difícil. HTML5 es una "marca" o un nombre que no se entiende realmente muy bien. Yo pertenezco al grupo de trabajo principal del estándar de HTML5 pero no puedo olvidar que no se hacen sitios web sin CSS, y otras API que se están desarrollando actualmente en otros grupos de trabajo. Lo que trataba de subrayar es que van a existir un número creciente de especificaciones adicionales.

Y creo que es importante que los desarrolladores tengan presente que no

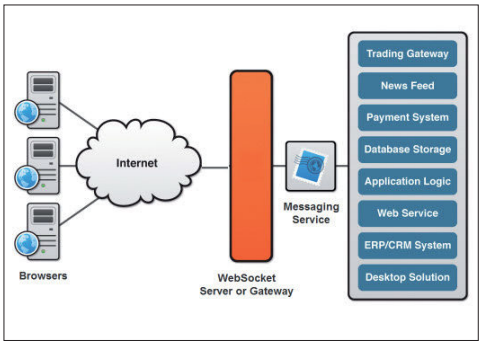


figura 12 Esquema de funcionamiento de Web Sockets (fuente: Websockets.org)

todas las especificaciones se hacen en la W3C. Un ejemplo perfecto de esto es Web Sockets, que es —probablemente— una de las partes más innovadoras de HTML5. Permite comunicaciones tipo dúplex entre cliente y servidor, mientras HTTP hoy es solamente del cliente a servidor, realmente. Pero el protocolo de trabajo con Web Sockets, se está realizando en ITF[22]. En W3C solo se trabaja en

la especificación del API.

Pero, incluso más importante, es lo que nosotros llamamos estabilidad. Y HTML 5, en el momento actual, no se ha llegado todavía a lo que llamamos "*Last Call*" (última llamada). Algunas partes ya son más estables, y ya se encuentran implementadas en distintos navegadores, pero otras son muy recientes. Así que pienso que lo realmente fundamental no es qué partes forman el estándar en su estado actual, sino lo estables que resultan hoy día.

Algunas veces, cuando no hay un consenso claro, la W3C tiene que decidir de acuerdo con algo que llaman la "Decision Policy". ¿Existe una votación? Y, de ser así, ¿cuán democrática resulta esa política de decisiones?

Es una pregunta interesante, porque, cuando los Ayuntamientos toman decisiones, por ejemplo, de que van a construir una nueva carretera, si hay una votación de 7 a 4, digamos, la propuesta se acepta. La W3C, en el espíritu de su fundador, Tim Berners-Lee, que sigue siendo hoy día el director, es una organización de consenso. Lo que significa tratar de encontrar la solución que produce el menor rechazo posible. Y esa política de decisión a la que te referías, es un intento de manejar un grupo de trabajo de más de 400 personas, de los que algunos son muy activos y otros no tanto.

Así que es importante para nosotros disponer de una política que establezca con claridad cómo tratar los *bugs*, y otros problemas detectados, y esa política nos lleva a solicitar cambios, renovaciones y otras soluciones alternativas. Y hacemos consultas *on-line*, a las que cualquiera puede responder. Y todos los rectores del grupo tenemos muy claro que —cuando buscamos el consenso— no contamos votos, sino que nos fijamos en la fuerza de los argumentos técnicos que se esgrimen.

De forma que, es posible, que alguien ofrezca una razón poderosa por la que alguna parte de la especificación deba ser abandonada, y que sea la única persona en hacerlo, pero si, como digo, sus razones están muy claramente razonadas, el grupo director lo va a tener muy en cuenta. En realidad, es una de las partes más difíciles de trabajar en W3C, y es una de las razones por las que somos 3 personas con diferentes historiales laborales y formativos.

Uds. disponen de una herramienta llamada Bugzilla, que supuestamente les ayuda a llevar el control de los bugs y otros errores que se van detectando. Esos errores son conceptuales,

[22] Se refiere a **Imec Technology Forum**, que días antes había celebrado una reunión internacional en Bélgica. Para más información ver http://www2.imec.be/be_en/education/conferences/itf2011.html

prácticos, o resultado de los test. ¿Cómo manejan este aspecto?

Sí, utilizamos una herramienta que se llama **Bugzilla**, para permitir que la gente pueda archivar los bugs. Y es parte de esa política de decisiones. Y está abierta a cualquiera, en realidad. Así que, cuando llegamos a la etapa Last Call, en una especificación, que es cuando ha sido ampliamente publicada y comentada, tratamos esa especificación como una beta de software.



Es muy fácil para cualquiera solicitar una cuenta para W3C, y poder dar de alta nuevos *bugs* con esa herramienta. Nosotros consideramos esos *bugs* como parte de la política de decisión. Uno de los 9 editores de las especificaciones de que consta el estándar, estará encargado de "tratar" el *bug* de forma adecuada. Algunos son muy fáciles de manejar (como los errores sintácticos), otros pueden estar solicitando una nueva característica, o cualquier otra cosa. La persona que envía el bug recibe un e-mail, indicando que el *bug* ha sido procesado.

Y se puede producir un diálogo, en el que una tercera parte opina igualmente sobre la resolución del *bug*, etc. Así que es un proceso bastante dinámico, y —en buena parte— asíncrono.

En la actualidad, yo diría que estamos gestionando una media de unos 200 y pico *bugs* por mes.

En la línea de la pregunta anterior, usted mencionó que buena parte de su tiempo lo pasa trabajando con la infraestructura de pruebas. ¿Cómo son realmente esas pruebas?

Uno de los objetivos principales del equipo es conseguir interoperabilidad. Ya sea con XML, XML-Schemas, XHTML o HTML5. Así que tenemos una política para manejar las pruebas, que resulta imprescindible antes de llegar a lo que llamamos Candidate Recommendation, y que comienza tras esa última llamada que mencioné antes. Y tenemos que coordinar todas las pruebas provenientes de todas las fuentes. Los fabricantes realizan pruebas individualmente, y cada uno tiene sus particularidades. Mozilla las hace de una forma y el grupo de Internet Explorer, de otra, por ejemplo.

Así que necesitamos algún tipo de infraestructura que nos permita aunar este conjunto de pruebas y operar de forma racional. Actualmente disponemos ya de más de 1.000 test aprobados. Y otros 20.000 más, que han llegado recientemente y que van a ser comprobados por nuestro "equipo de *testing*"[23], ya que disponemos de un equipo dedicado exclusivamente a estas tareas.

Así que aprovecho la ocasión para invitar a cualquier desarrollador Web a que entre en la página oficial de la W3C, busque la página de *testing*, y pruebe y envíe los resultados de sus propios test si lo considera interesante. Personalmente, no me extrañaría que, dentro de un año, cuando empecemos a recibir las pruebas de las primeras implementaciones en navegadores reales, alcancemos una cifra cercana a los 75.000 test realizados. CSS 2.1, que está a punto de ser terminada por la W3C tiene unos 9.000 test realizados.

Vamos a movernos, si le parece, a terrenos más prácticos y relacionados con su presencia aquí. ¿Cuál piensa Ud. que es el mensaje principal que trae en esta visita a España y cuáles son los puntos principales que piensa exponer esta tarde en su presentación?

Creo que tengo dos mensajes. Uno, por mi pertenencia al W3C y otro por representar a Microsoft. La W3C está convencida que esta Plataforma Web Abierta, que incluye HTML 5, CSS, las API relacionadas, ECMAScript -que se está renovando en ECMA—, el protocolo que está siendo desarrollado por ITF, etc., son tecnologías que cambiarán las reglas del juego.

Y piensa que nos va a permitir sitios Web muy innovadores y potentes, y también creemos que vamos a ver un gran movimiento hacia las aplicaciones HTML 5. Y es posible que —dentro de 2 ó 3 años— esas aplicaciones puedan conseguir resultados tan interesantes como las llamadas aplicaciones nativas, ya se trate de plataformas PC o de dispositivos móviles, teléfonos o *tablets.* Así que ese mi principal mensaje. HTML 5 es muy importante, y la W3C es el sitio donde se está realizando ese trabajo y el *Advisory Comitee Meeting*, al que he asistido en Bilbao es otro ejemplo de reunión de la comunidad en torno a una misma idea.



Así que hemos visto como compañías que antes no estaban en estas reuniones, como Disney, LG o Sony, han vuelto a participar activamente. Lo mismo sucede con las compañías telefónicas, como Nokia, Eriksson o AT&T. Otro ejemplo es Comcast, una de las compañías de cable más importantes de EE.UU., que ahora es miembro de la W3C.

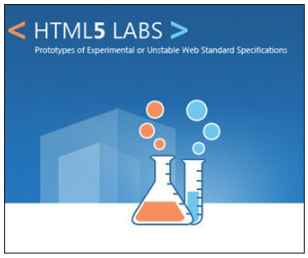
Desde el punto de vista de Microsoft, he estado trabajando con el equipo de Internet Explorer durante unos dos años, desde IE8, luego IE9 y ahora con la *preview* de IE10. Y creo que la implicación de Microsoft con el estándar es muy

[23] *Testing Task Force*, en el original inglés.

fuerte en todos los sentidos, al igual que lo están haciendo ECMA, e ITF con otras tecnologías relacionadas.

Pero lo más importante para Microsoft, no es hacer una "carrera" para ver quién lo implementa primero, sino fijarse en la estabilidad, y de cara a nuestros clientes, asegurarnos que no implementamos las especificaciones hasta que no sean lo suficientemente estables como para ofrecer un camino seguro a los desarrolladores Web.

Así que, animamos a la comunidad de programadores a que prueben nuestras implementaciones, que solemos actualizar con una frecuencia de 10/12 semanas, o que entren en sitios como HTML5 Labs[24], donde se están construyendo prototipos de aspectos como IndexDB que son de esperar que veamos en Internet Explorer y en otros navegadores. Consiste en una base de datos de cliente, muy sencilla, solamente parejas "Clave-Valor" (*Key-Value Pairs)*, que permite muchas implementaciones en el cliente.



Así que, desde nuestro punto de vista, es bueno que quede claro que nos importa la estabilidad del producto ante todo.

¿Nos puedes decir algo sobre las expectativas de adopción del estándar que tiene Microsoft, y cuánto tiempo calcula que va a llevar el proceso?

Depende de la tecnología que lo aplique. Por ejemplo, la presencia de LG o Sony es un indicativo de que ellos piensan incluir un navegador en sus televisores. Y hay compañías de automóviles que están pensando cómo adaptar sus mantenimientos, sus sistemas de seguridad, etc. Así que una vez más esperamos que HTML 5 sea el aglutinante de muchas tecnologías para muchos sectores distintos de la industria.

El programador en general, cuando oye hablar de HTML 5 no puede evitar pensar en JavaScript. Y muchos ya afirman que esto supone una vuelta a los viejos tiempos, o sencillamente, un paso atrás en las capacidades de desarrollo. ¿Qué piensa Ud. al respecto?

Es interesante. Como indique en una respuesta anterior, me considero "darwiniano" en lo referente a especificaciones. Tenemos especificaciones buenas, y otras normalitas. Pero lo que resulta realmente importante es la adopción. Y creo que la

[24] http://blogs.msdn.com/b/mvplead/archive/2011/06/10/lt-html5-labs-gt.aspx

prevalencia de la Web, y el movimiento hacia tener cada vez más aplicaciones web, está haciendo que esas aplicaciones sean cada vez más potentes.

Y al objeto de poder aprovechar todas las posibilidades de una plataforma para una aplicación, realmente se necesita tener acceso a esa plataforma. Por ejemplo, la geo-localización en los móviles. O el audio, el vídeo o las cámaras. Y el acceso a todas esas posibilidades se está haciendo mediante diversas API de JavaScript.

Así que las herramientas que tenemos hoy, ya sea Visual Studio o Eclipse, se van a tener que adaptar y ayudar mucho más al desarrollador a utilizar esas API. Todavía no hemos llegado a un grado de soporte suficiente, pero estoy convencido que ese es el objetivo de la siguiente generación de herramientas de desarrollo.

¿El futuro de la Web rechazará los "plug-in" de cualquier tipo? ¿Se pretende que los navegadores prescindan de componentes extra de cualquier clase?

Yo no pienso que aplicaciones como Flash o Silverlight vayan a desaparecer en absoluto. Volviendo a lo que decía antes, creo que estas plataformas son muy populares debido las herramientas de desarrollo, que ayudan a los programadores a construir esos complementos. Así que, hasta que no existan las herramientas necesarias para que HTML 5 pueda competir con esos complementos (*add-ons*) vamos a continuar viendo soporte de Flash por parte de Adobe, y soporte de Silverlight por parte de Microsoft.

Hay muchas aplicaciones verticales para los que estas tecnologías son las más apropiadas. Pero también creo que con el tiempo comenzaremos a ver cada vez más aplicaciones basadas en HTML 5 y a más productores de navegadores hacer lo que Microsoft está haciendo con IE, que es conectar la tecnología que está en el navegador directamente al hardware del equipo.



Por ejemplo, utilizando el procesador gráfico para obtener un vídeo más suave, y mejores animaciones, para conseguir lo que ahora se llaman "Juegos para PC", directamente en HTML. Pero todavía nos queda por recorrer un largo camino. La tecnología nos ayudará y las herramientas de desarrollo también y por supuesto, ¡acabar la especificación nos ayudará también!

Lo que el desarrollador necesita es escribir una sola aplicación para cualquier navegador.

¿Y qué sucede con ECMAScript 5? Tengo entendido que no se construye en W3C. ¿Existe algún tipo de coordinación entre ambos grupos?

Cuando lanzamos IE9, incluimos un nuevo motor JavaScript en el producto. Nues-

tra división de interoperabilidad implica a diferentes grupos en Microsoft. Nuestro equipo de JavaScript pertenece a la división Server and Tools, mientras que Internet Explorer pertenece a la de Windows. Cuando sacamos IE, combinamos las dos tecnologías. Y de hecho, mi equipo es la tercera parte de este juego, porque tenemos la responsabilidad dentro de Microsoft de todo lo relacionado con la W3C.

Así que trabajamos juntos, y yo mismo he estado en varias reuniones con los equipos que están haciendo ECMAScript en la actualidad. Pero es que los equipos de desarrollo de TC39[25] también se reúnen periódicamente para resolver problemas técnicos de la implementación. Y lo mismo tienen que hacer los de otras empresas si quieren que la implementación del estándar sea adecuada.

¿Qué significa exactamente el término HTML nativo, en palabras de Dean Hachamovitch?

Creo que a lo que el jefe de desarrollo de IE9 se refería es al aprovechamiento del hardware de la plataforma por parte del navegador. Al hecho de que el navegador "conoce" la plataforma en la que se está ejecutando, y hace uso de las capacidades que esta plataforma le ofrece. Y analizar el sistema operativo hasta el nivel de hardware, de forma que pueda utilizar la GPU para acelerar la experiencia y acelerar las respuestas del sitio Web o la aplicación que se está mostrando al usuario.

Uno de los problemas relacionados con las aplicaciones tiene que ver con las API relacionadas con el estándar. Podemos usar Modernizr para que los antiguos navegadores "comprendan" HTML 5. Pero, ¿qué sucede con las API necesarias para el desarrollo, como Local Storage, Session Storage, File API, WebSQL, Web Sockets, Office Apps…?

Algunas de esas especificaciones entran en categorías muy distintas. Por ejemplo, WebSQL creo que es una "especie en extinción". Y la mayoría de los fabricantes, reconocen que la especificación necesaria para contar con una base de datos en el cliente va a ser IndexDB. Y de hecho, el trabajo con esa especificación dentro de la W3C ya no continúa. File API, desde nuestro punto de vista está llegando a la categoría de estable, y tenemos un prototipo en HTML 5 Labs, y creo que podremos verlo en IE en algún momento futuro. Y lo mismo sucede con las otras especificaciones que men-



[25] Para más información sobre el grupo TC39 ECMAScript, ver http://www.ecma-international.org/memento/TC39-M.htm

cionabas que pienso que van a seguir madurando e incorporándose paulatinamente.

Web Sockets, es uno de los buenos ejemplos de esas especificaciones. Es una de las más prometedoras, y eso es lo que respondo cuando me preguntan sobre ella, aunque también les digo lo mismo cuando me preguntan por cuál es el API más "peligrosa"… Nosotros trabajamos con la ITF (como dije, la entidad que gestiona los protocolos), aunque la parte del API la seguimos llevando nosotros. Pero el protocolo está ahora en su versión 7. Y eso que llevan trabajando en ello unos 7 meses. De hecho, el protocolo ha cambiado tanto desde su inicio que, ahora la especificación del API que nosotros llevamos tiene que estar alineada con el nuevo protocolo. Ese trabajo está pendiente también.

Así que, desde el punto de vista de Microsoft, no vamos a implementar esa especificación hasta que consideremos que es lo suficientemente estable. La lista que comentabas tiene especificaciones que se encuentran en distintos estados de madurez, pero estoy convencido de que la mayoría de las que mencionabas estarán disponibles dentro de un año cuando lleguemos al estado de Candidate Recommendation.

¿Qué plataforma de Microsoft piensa Ud. que está más orientada a futuro: Web Matrix (con Razor) o ASP.NET?

Normalmente, cuando me preguntan por tecnologías, siempre respondo que lo mejor es que cada uno escoja aquella que conoce mejor. Y estoy convencido de que es más importante comprender bien la tecnología, que el hecho de usar una u otra plataforma o herramienta.

No creo que haya una sola respuesta aquí. Hay muchos entornos de trabajo interesantes que la gente puede utilizar, para la construcción de aplicaciones web, pero, si pensamos en el éxito de los proyectos, casi siempre se trata de desarrolladores que conocen muy bien las plataformas y herramientas en las que desarrollan. Así que recomiendo que la gente llegue a un nivel alto de conocimientos en una o dos, de forma que puedan brillar en lo que hacen.

Y finalmente, ¿cuál es el papel de Silverlight 5, (aparte de Windows Phone 7) si es que tiene alguno?

No pienso que tecnologías como Silverlight vayan a desaparecer en un futuro. Creo que hay muchas aplicaciones en las que —desde hace ya algún tiempo— Silverlight es la tecnología más adecuada a utilizar.

Por otro lado, HTML 4 disponía de la etiqueta `<object>` que nos permitía

incluir cosas como vídeo y audio. HTML 5 dispone de etiquetas **<video>** y **<audio>** así que, los *add-ons* eran extensiones necesarias, que —en este aspecto— ya no lo son. Pero la etiqueta **<object>** no va a desaparecer, eso es seguro. Y la gente va a continuar utilizándola.

Creo que lo que sucederá tendrá que ver con las mejores herramientas y con las mejores experiencias, y con la amplitud de la cobertura que se consiga con ellas. Será una cuestión de sopesar los factores de cobertura que la tecnología ofrece respecto a los beneficios obtenidos con otra tecnología. Así que las diversas situaciones harán que unos vayan por un camino y otros por otro.

Pero, en el caso de Silverlight, estoy seguro de que existen patrones muy fuertes que avalan su utilización, y eso supondrá que la gente va a continuar utilizándolo.

Una de las razones por las que Microsoft incluyó IE9 en Windows Phone fue esa idea de la cobertura ampliada. Y que los usuarios y desarrolladores apreciasen esa independencia de funcionamiento, tanto si están en Windows, como en Windows Phone. De hecho, estamos viendo la aparición de *start-ups*, cuya idea principal es aprovechar este estándar HTML 5, para construir aplicaciones.

Pues muchas gracias por sus palabras, y esperamos que su estancia en España le resulte agradable y podamos verle pronto por aquí.

Ha sido un placer venir aquí. Y había estado en España un par de veces antes. El clima es excelente.

Para concluir quería añadir que, cuando **Tim Berners-Lee** creó la W3C, su objetivo era hacer que la Web tuviese un impacto mundial, y creo que HTML 5 es otro ejemplo de caso de éxito de la W3C. Tenemos soporte de un número creciente de miembros, y todos ellos están muy interesados en la construcción de esta "Plataforma Web Abierta".

Referencias

Holguera, Javier (2011), "Internet Explorer 9 y HTML 5". Netalia, p. 15.
Conociendo RDF y la Web Semántica con N3, **http://www.w3.org/2000/10/swap/Primer**
Hoja de Ruta de la Web Semántica, **http://www.w3.org/DesignIssues/Semantic**
¿Que es la Web Semántica?, **http://purl.org/swag/whatIsSW**
Semantic Web Primer, **http://uwimp.com/eo.htm**
Introducción a la Web Semántica–Extenso, **http://logicerror.com/semanticWeb-long**
SciAm: *The Semantic Web*, **http://www.scientificamerican.com/2001/0501issue/0501berners-lee.html**
Construyendo la Web Semántica, **http://www.xml.com/pub/a/2001/03/07/buildingsw.html**
The Semantic Web, Taking Form, **http://infomesh.net/2001/06/swform**
SW Activity Statement, **http://www.w3.org/2001/sw/Activity**
SWAD, **http://www.w3.org/2000/01/sw**

Referencias de la Web Semántica

"W3C Semantic Web Frequently Asked Questions". W3C. Retrieved March 13, 2008.

Berners-Lee, Tim; **James Hendler** and **Ora Lassila** (May 17, 2001)."The Semantic Web". *Scientific American Magazine*. Retrieved March 26, 2008.

Herman, Ivan (March 12, 2008). "W3C Semantic Web Activity". W3C. Retrieved March 13, 2008.

Berners-Lee, Tim; **Fischetti, Mark** (1999). *Weaving the Web*.HarperSanFrancisco. chapter 12. ISBN 978-0-06-251587-2.

Gerber, AJ, **Barnard, A & Van der Merwe, Alta** (2006), A Semantic Web Status Model, Integrated Design & Process Technology, Special Issue: IDPT 2006

Gerber, Aurona; **Van der Merwe, Alta**; **Barnard, Andries**; (2008), A Functional Semantic Web architecture, European Semantic Web Conference 2008, ESWC'08, Tenerife, June 2008.

Karger, David. "What Would It Mean to Blog on the Semantic Web". Retrieved November 15, 2010.

Victoria Shannon (June 26, 2006). "A 'more revolutionary' Web".*International Herald Tribune*. Retrieved May 24, 2006.

"The advent of Web 3.0". *Semantic Web Company*. Retrieved November 14, 2010.

Conrad Wolfram on Communicating with apps in web 3.0 IT PRO, 17 Mar 2010

Shah Newaz Alam. "Web 3.0 vs Web 2.0". *Buzzle.com*. Retrieved November 14, 2010.

Steve Spalding (July 14, 2007). "How To Define Web 3.0".*bowtosplitanatom.com*. Retrieved November 14, 2010.

Artem Chebotko and **Shiyong Lu**, "Querying the Semantic Web: An Efficient Approach Using Relational Databases", LAP Lambert Academic Publishing, ISBN 978-3-8383-0264-5, 2009.

Gärdenfors, Peter (2004). *How to make the Semantic Web more semantic*. IOS Press. pp. 17–34.

Timo Honkela, **Ville Könönen**, **Tiina Lindh-Knuutila** and **Mari-Sanna Paukkeri** (2008). "Simulating processes of concept formation and communication". *Journal of Economic Methodology*.

Ivan Herman (2007). "State of the Semantic Web". *Semantic Days 2007*. Retrieved July 26, 2007.

Berners-Lee, Tim (May 1, 2001). "The Semantic Web". Scientific American. Retrieved March 13, 2008.

Nigel Shadbolt, Wendy Hall, Tim Berners-Lee (2006). "The Semantic Web Revisited". *IEEE Intelligent Systems*. Retrieved April 13, 2007.

Lee Feigenbaum (May 1, 2007). "The Semantic Web in Action".Scientific American. Retrieved February 24, 2010.

Martin Hilbert (April, 2009). "The Maturing Concept of E-Democracy: From E-Voting and Online Consultations to Democratic Value Out of Jumbled Online Chatter". Journal of Information Technology and Politics. Retrieved February 24, 2010.

http://policyawareweb.org/

"RDF tutorial". Dr. Leslie Sikos. Retrieved 2011-07-05.

"Resource Description Framework (RDF)". World Wide Web Consortium.

"Standard websites". Dr. Leslie Sikos. Retrieved 2011-07-05.

Allemang, D., Hendler, J. (2011). "RDF—The basis of the Semantic Web. In: Semantic Web for the Working Ontologist (2nd Ed.)". Morgan Kaufmann.

Lukasiewicz, Thomas; **Umberto Straccia**. "Managing uncertainty and vagueness in description logics for the Semantic Web".

See, for instance: **Bergman, Michael K.** "Sweet Tools". *AI3; Adaptive Information*, *Adaptive Innovation*, *Adaptive Infrastructure*. Retrieved January 5, 2009.

"GoodRelations: The Web Ontology for E-Commerce". E-Business + Web Science Research Group.

"GoodRelations Project Main Page".

Hepp, Martin (September 29 — October 3, 2008). "GoodRelations: An Ontology for Describing Products and Services Offers on the Web".*Proceedings of the 16th International Conference on Knowledge Engineering and Knowledge Management (EKAW2008)*.

capítulo 2

Herramientas y depuración

Múltiples herramientas

Aunque existen una infinidad de herramientas para crear y editar los tipos de código fuente que asociamos con el estándar (los 3 formatos) nos vamos a centrar en las herramientas de Microsoft y en las propias de los navegadores: herramientas de seguimiento, análisis y gestión del código, control de estilos, etc., que los fabricantes van renovando periódicamente con cada actualización.

En cuanto a herramientas de desarrollo, y en especial en Visual Studio, disponemos de editores de todos los archivos relacionados con HTML 5: Soporte de código de marcado HTML 5 y XHTML 5, soporte de archivos CSS, y de JavaScript, con posibilidades de depuración nuevas y otras mejoradas. En especial, la edición Visual Studio 2012, ya disponible, cuenta con muchas características avanzadas que utilizaremos en este libro.

Independientemente, podemos utilizar otras herramientas, como es el caso de **Fiddler**, que cuenta con capacidades muy especializadas, sobre todo para el análisis del comportamiento de las páginas, tiempos de carga, tráfico de red, simuladores, etc.

Las herramientas de los navegadores

Dentro de los navegadores más populares siempre nos encontramos con un apartado de "Herramientas del Desarrollador", cosa que sucede con todos los que utilizaremos aquí: IE10, Firefox, Chrome, Opera y Safari. Comenzamos por un breve repaso de lo más característico en cada una de ellas.

La herramienta de desarrollo de Internet Explorer (F12)

En el caso de Internet Explorer (y también para algunos otros), se accede a la herramienta de desarrollo mediante la tecla F12. Esto lanza una ventana separada donde se tiene acceso al código fuente (HTML, CSS y JavaScript), y a un amplio menú de opciones para desarrolladores, que incluyen marcadores dinámicos de elementos, selectores, la posibilidad de deshabilitar elementos y ver el resultado (CSS, imágenes, *scripting*), rastreadores de estilos, mecanismos visuales para analizar el *boxing* de cada elemento (la distribución visual de cada rectángulo generado por él), manejo de cachés y *cookies*, herramientas de validación de código, manipuladores de imágenes, visores de meta-información, etc.

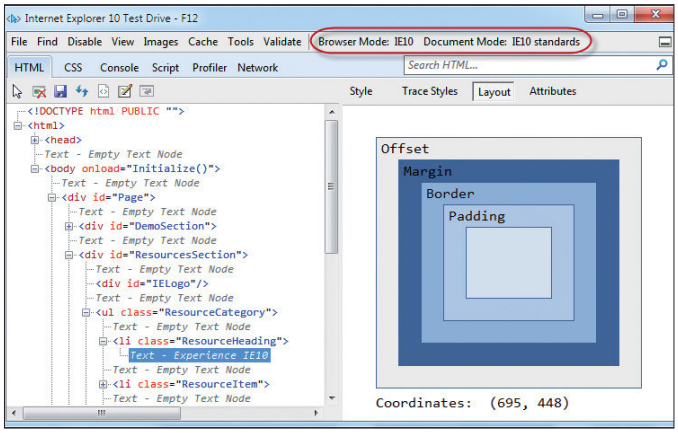


figura 1 La herramienta de desarrolladores de IE10

Como vemos en la figura 1, siempre tenemos disponible el código fuente de la página, así como menús complementarios para opciones de manejo de hojas de estilo (CSS), una consola, y seguimiento del código JavaScript como las más interesantes. Hay otras adicionales, como el Generador de perfiles y la herramienta de Red, que habilita opciones para el seguimiento del tráfico, y permiten estudiar los tiempos de carga de cada elemento de la página, y determinar cuellos de botella y otras características.

También dispone de una opción "Browser Mode", para poder simular el comportamiento con motores anteriores de Internet Explorer.

Otra característica importante del depurador es la capacidad de analizar el código

JavaScript, pudiendo insertar puntos de ruptura y estudiar el estado de las variables relacionadas. Incluso podemos establecer condiciones de ruptura al estilo que es habitual en las herramientas de desarrollo, como vemos en la figura 2.

Además, y de forma similar a Visual Studio, dispone de inspecciones de variables

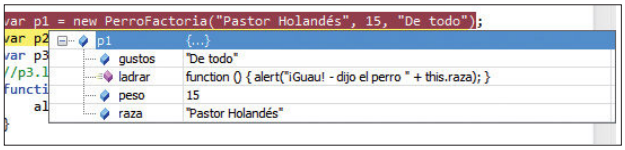


figura 2 Inspección de variables en el depurador de IE.

en línea, con lo que podemos comprobar cualquier valor en tiempo de ejecución, sin tener que establecer previamente un punto de observación sobre una variable (se muestra en la figura 3):

Y, por supuesto, podemos analizar todo lo relacionado con los encabezados de las

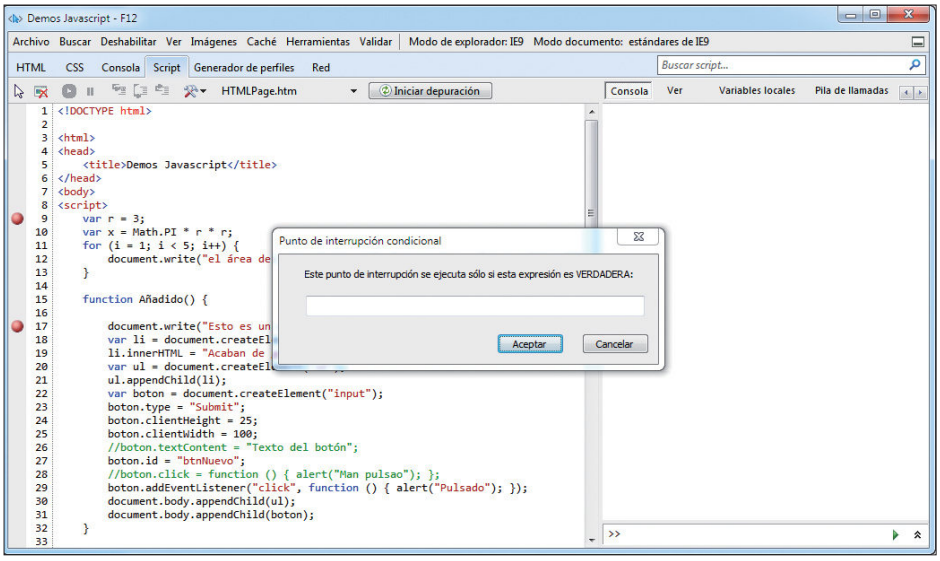


figura 3 El depurador con puntos de ruptura.

peticiones Web, tanto en las solicitudes como en las respuestas.

También merece la pena mencionar las opciones del menú "Ver". Cuando alguna de ellas se activa, la página queda "redecorada" con las indicaciones de los elementos activados, como puede ser la información del ID y clase de elemento, o los vínculos de cada uno, que muestra la Figura 4, depurando un fragmento de una página en Twitter.

Y no solo disponemos de la opción de cambiar el motor de interpretación. En el

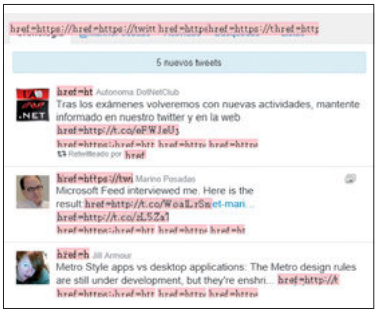


figura 4 Resultado de activar el menú "Ver/Rutas de Acceso al Vínculo".

menú de herramientas se nos permite simular el comportamiento de la página en otros navegadores y con otras resoluciones. También es posible deshabilitar la ejecución de *scripts* o la aplicación de estilos CSS, para ver más claramente el efecto resultante de las variantes "con" y "sin" tales características en una página dada. Recomendamos al lector que lo pruebe, ya que la experiencia puede ser sorprendente, según qué página se visualice.

Otra opción interesante (aunque no es nueva), consiste en habilitar la opción "Buscar/Seleccionar elemento con un clic", con la que podemos seleccionar manualmente en la página un elemento del que queremos información, y —de forma automática— el explorador HTML se desplazará al elemento correspondiente en la jerarquía del código fuente, mostrándonos todo lo relacionado con él, como muestra la imagen de la figura 5.

Para finalizar este pequeño repaso a la herramienta de desarrolladores de IE, cabe mencionar que también es posible generar informes por página, relacionando diversos aspectos de ella, como los gráficos que contiene, su accesibilidad, y su nivel de validación del código fuente respecto a un estándar concreto (para ello, envía la página al servicio de validación de marcado de la W3C).

En lo que concierne al análisis de carga de los recursos de la página, tenemos, en el apartado "Network", un buen conjunto de opciones para monitorizar este tráfico y controlar cada elemento cargado por la página. Concretamente, podemos tener una visión global del retardo originado por los elementos al cargar, pero también disponemos

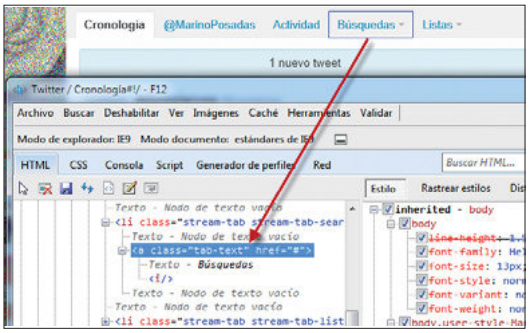


figura 5 Buscador rápido de elementos.

de la opción de analizar individualmente cada uno, como se aprecia en la siguiente figura que captura un tráfico habitual de Twitter:

No obstante, si nuestra depuración precisa de funciones más avanzadas relacionadas con el tráfico de red, nuestra re-

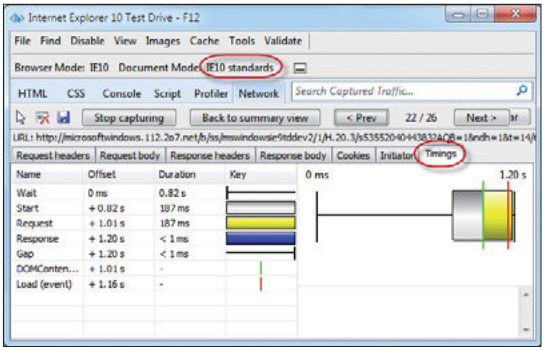


figura 6 Analizador de tráfico de red en la herramienta de desarrollo de IE10.

comendación es complementar el uso de F12 con otra herramienta gratuita de Microsoft: **Fiddler**.

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**, Program Manager de Mi-

crosoft ha estado desarrollando de forma independiente durante años. Merecería un tratamiento aparte, pero, por razones de espacio, vamos a explicar aquí solo lo más importante de su funcionamiento y características.

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 web[1] 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.

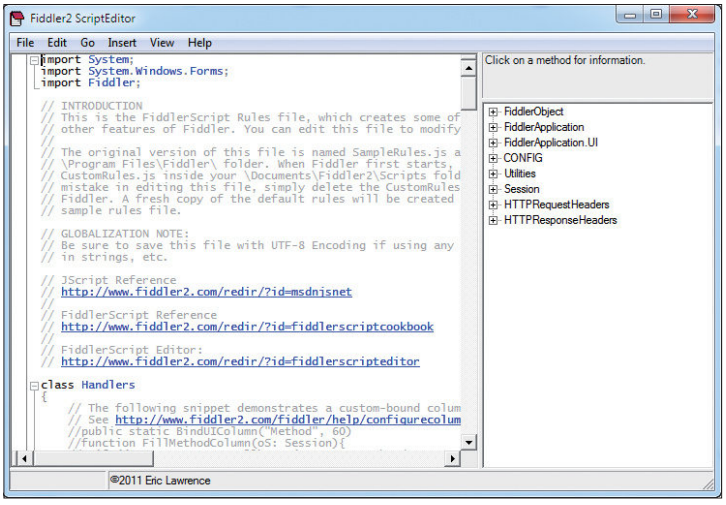


figura 7 El Editor de Scripts para el modelo de objetos de Fiddler.

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

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 8):

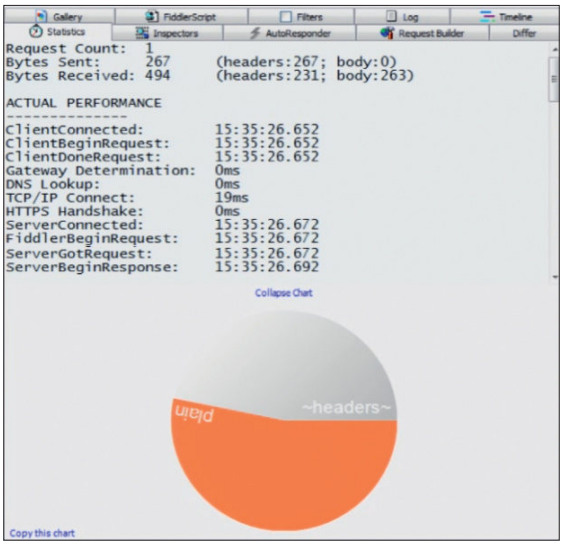


figura 8 Informe estadístico de Fiddler 2.

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 tráfico de este tipo. La figura 9 muestra la ventana de creación de un certificado con este propósito.

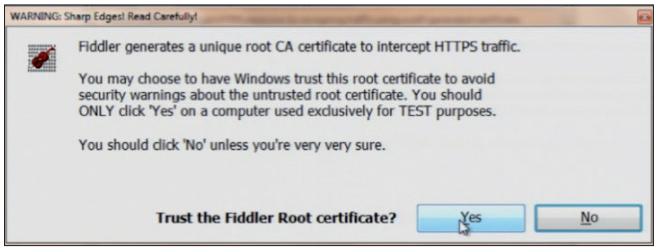


figura 9 Creación de un certificado para habilitar la monitorización de tráfico seguro.

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 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:

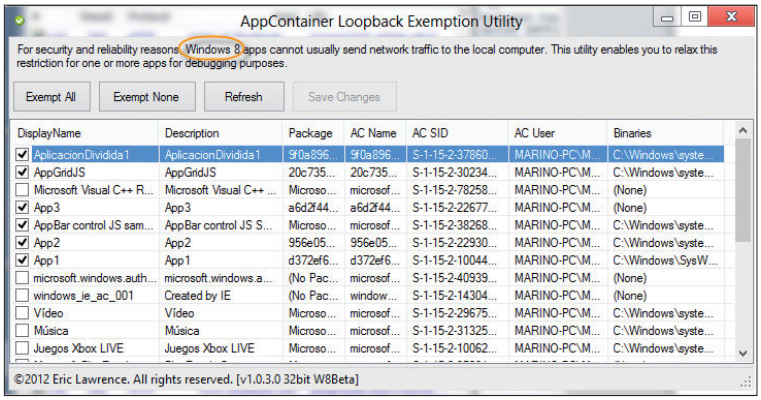


figura 10 Utilidad de configuración de control de tráfico en aplicaciones Windows 8.

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.

Una ventaja notable es que, como proxy, puede funcionar con cualquier cosa que

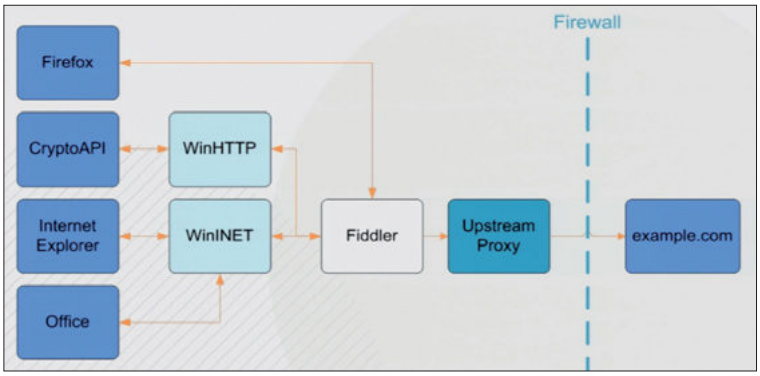


figura 11 Esquema del escenario de trabajo de Fiddler.

lo soporte, como pueden ser aplicaciones, por lo que no se limita al tráfico de los navegadores.

La ventana principal, siempre presenta el trafico 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.

Comparativa de capacidades de análisis de tráfico de red entre F12/IE y Fiddler

Para darnos cuenta de qué capacidades ofrece Fiddler respecto a los otros analizadores de tráfico, como el de F12, podemos considerar una tabla como la tabla 1.

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:

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).

Herramienta F12 (Solapa Red)	Fiddler
Muestra caché y solicitudes	Muestra y modifica solo las solicitudes
Muestra las descargas del proceso actual	Muestra tráfico de todos los procesos
Límite de 6 conexiones por host	Límite de 12 conexiones por host
Muestra trafico descifrado de HTTPS técnica "man-in-the-middle"	Descifra el tráfico HTTPS mediante una
Excelente formateador de código JavaScript	"Plug-in" para formato de código JavaScript (algo menos potente)
Detección de contenido mixto muy explícita	Detección de contenido mixto menos explícita
Exporta la información a `NetworkData.xml`	Importa información de `NetworkData.xml`

Tabla 1: comparativa funcional de capacidades de captura de tráfico

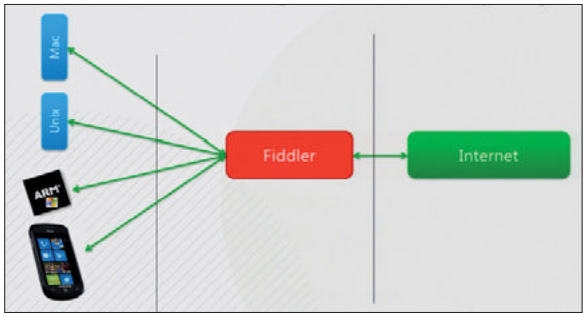


figura 12 Esquema del soporte de dispositivos y plataformas de Fiddler.

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 13), que nos da un resumen (texto e imagen) de los porcentajes de utilización de cada recurso y el tiempo que han tardado en cargarse.

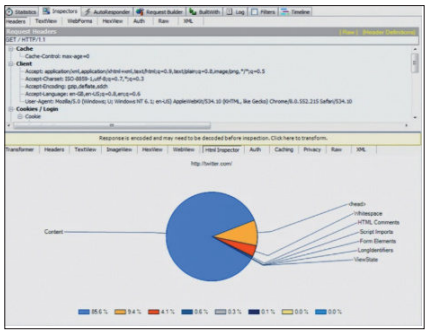


figura 13 HTML Analyzer Inspector de Fiddler.

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 14.

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.

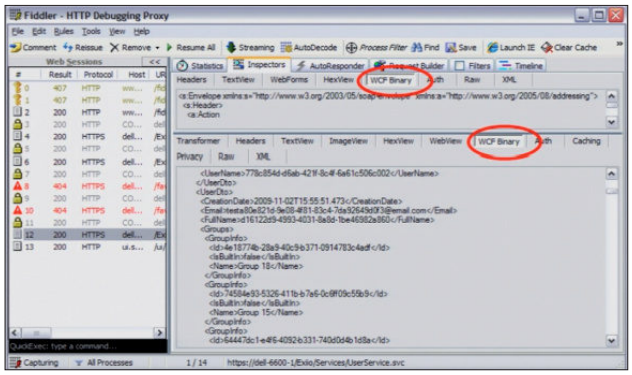


figura 14 *WCF Binary Inspector* en funcionamiento.

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 15:

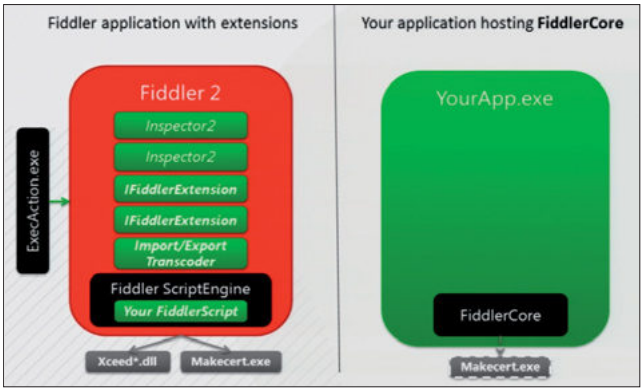


figura 15 EL modelo de objetos programable de Fiddler.

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

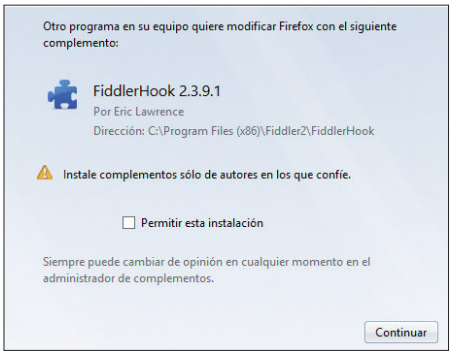


figura 16 FiddlerHook: proceso de instalación en FireFox.

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):

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.

FireBug para FireFox

Otra herramienta muy popular y de excelentes prestaciones es **FireBug**, la herramienta

del desarrollador de FireFox. En principio, se presenta como un complemento actualizable de fácil instalación, y 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 (ver figura 17).

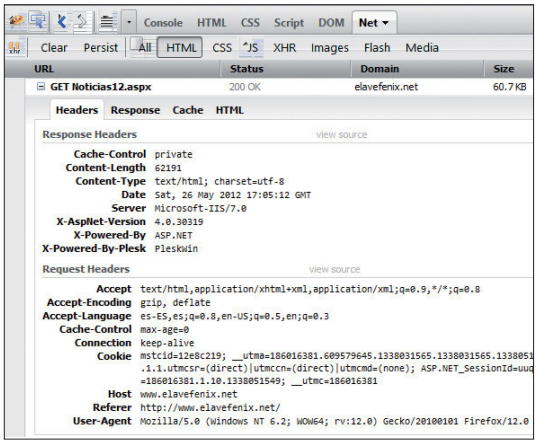


figura 17 Ventana que muestra FireBug una vez activada mostrando el submenú asociado a procesos de red.

Como vemos, es similar al de Internet Explorer (aunque se echa de menos alguna opción 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 la opción ("Click an element to inspect"), similar a la de IE10, que permite seleccionar cualquier elemento de la página, haciendo que FireBug localice el elemento en la ventana HTML y, una vez activado éste mediante otro clic, usar la ventana derecha del complemento para analizar todo lo que se aplique a ese elemento: estilos, JavaScript,

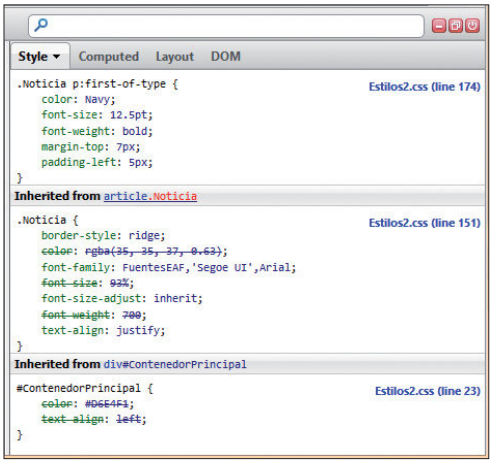


figura 18 Ventana derecha de propiedades de un elemento seleccionado en FireBug

ventana de DOM (y *layout* correspondiente), tiempo de carga (si se trata de un recurso empotrado), etc. Lo podemos ver en la Figura 18:

En general, podemos decir que casi todas las herramientas de navegador son bastante similares, y ofrecen un conjunto común de herramientas muy completo.

Vistas 3D de cualquier página

Como cada herramienta de navegador tiene una estrategia, a veces nos encontramos con aspectos peculiares de una herramienta que la hacen especialmente interesante, y esto es lo que sucede con la Vista 3D.

Es una de las opciones que más no ha llamado la atención de la última versión FireBug y se trata de "reinterpretar" la página de forma visual, de tal forma que podamos apreciar con un vistazo sus elementos, como si todos ellos tuviesen una dimensión en el eje **z** (el que apunta al espectador), de forma que aparecen montes y valles en función del número de elementos contenedores y contenidos, como si las páginas tuvieran pers-

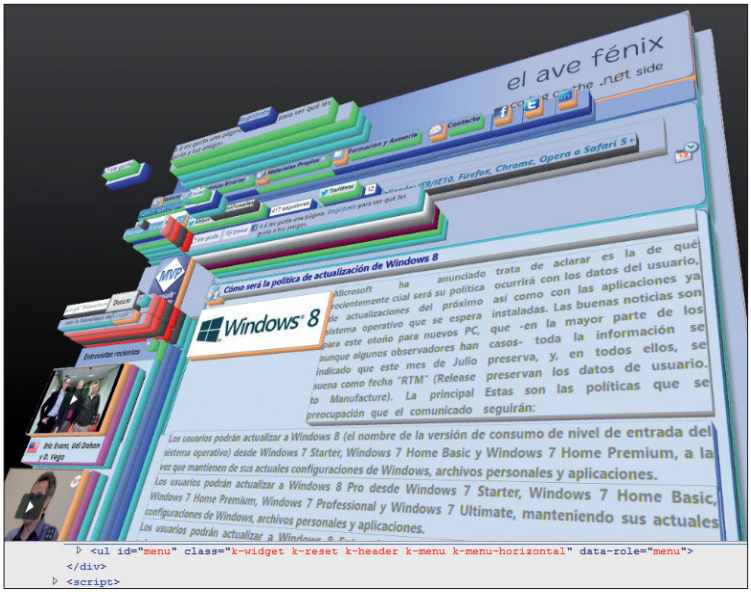


figura 19 Opción "Vista 3D", del depurador FireBug.

pectiva hacia el usuario. Esta opción se hace disponible al seleccionar "Inspeccionar elemento", en el menú "Desarrollador Web". Es mejor verlo en imagen como la que muestra la figura 19:

Y, naturalmente, la lista de opciones disponibles es muy grande y una de las más

completas que hay en los navegadores actuales. Además, las herramientas de evaluación de código nos indicarán cualquier incongruencia encontrada.

Las herramientas de desarrolladores de Google Chrome

Al igual que los anteriores, Chrome ofrece (como parte de sus menús, o mediante la tecla F12, sin necesidad de instalación), una opción que nos abre una ventana similar a las que hemos comentado antes, con parecidas opciones, y un tratamiento muy completo de todas ellas.

Por ejemplo, podemos depurar código JavaScript, tanto si se trata de *scripts* empotrados en la página, como si es un archivo separado del código fuente, y podemos

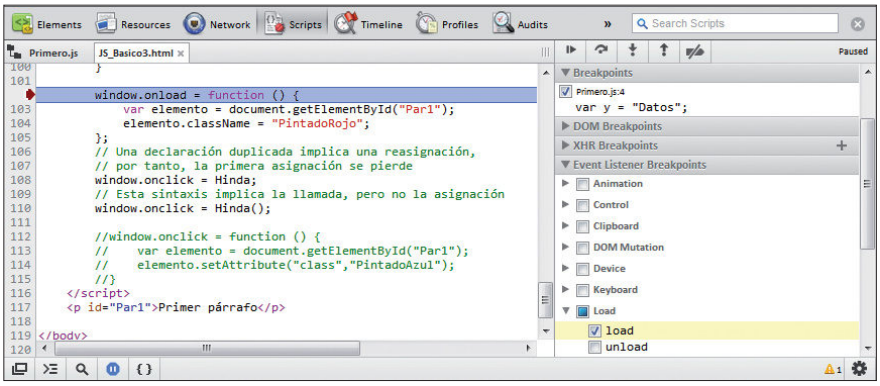


figura 20 Herramientas de desarrollador de Chrome, mostrando un punto de ruptura en el evento *onload* de una página.

poner puntos de ruptura en modo similar a como vimos en el apartado de IE10, controlando eventos predeterminados o inspeccionando una línea de código en particular, como se aprecia en la figura adjunta:

Recordemos que estas opciones de depuración propia de los navegadores pueden sernos muy útiles cuando determinado elemento parece no funcionar en un agente de usuario concreto, o cuando queremos garantizar que una cierta característica del estándar está soportada por ese navegador.

Otra utilidad interesante consiste en poder comprobar excepciones generadas por librerías JavaScript en el contexto del DOM (las llamadas DOM Exceptions). Una de las vistas de la ventana de "Breakpoints", nos ofrece información detallada de esta función como se ve en la figura 21.

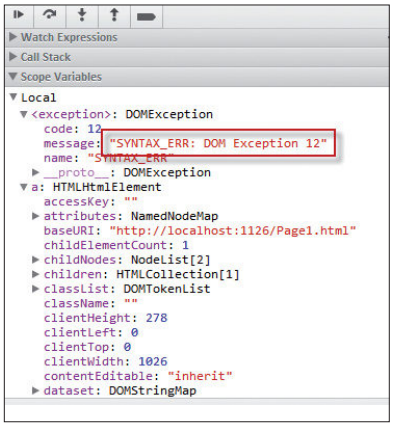


figura 21 Detalle de la ventana "Breakpoints" del depurador de Chrome.

Y ya que esto está dedicado al soporte del estándar HTML5, hay que decir que este depurador incorpora analizadores para algunas de sus API, como Local Storage, Session Storage, Application Cache y Bases de Datos definidas mediante IndexedDB

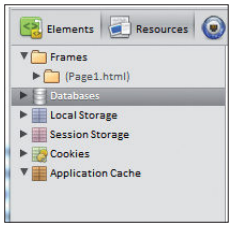


figura 22 Opciones del menú *Resources* en la herramienta de desarrollador de Chrome

(veremos estos aspectos en el último capítulo). El gráfico de la figura 22 muestra el menú "Resources" de esta herramienta con las diferentes opciones:

Al igual que las otras herramientas comentadas, cuenta con un mecanismo configurable de perfiles ("Profiles") para controlar el tiempo de CPU que consume cada recurso, pudiéndose controlar independientemente aquellos que tienen que ver con *scripts* y los que están relacionados con hojas de estilo. 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 ejecución. Cuando la carga se completa, basta con volver a esta opción y comprobar el informe generado, como vemos en la figura 23.

Y hablando del soporte del nuevo estándar, los objetos *WebWorker* (o simplemente,

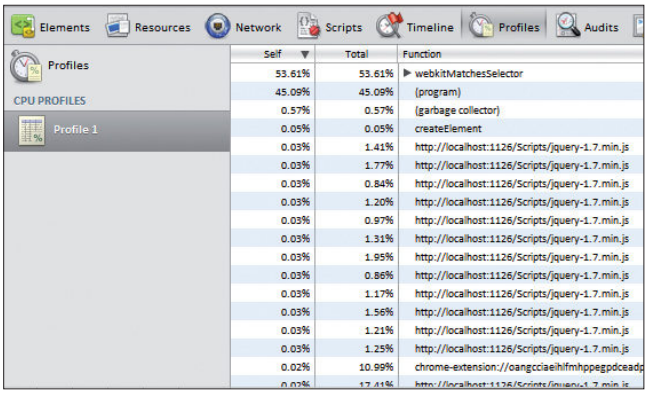


figura 23 Informe de la herramienta de desarrolladores de Chrome generado por la opción "Profiles"

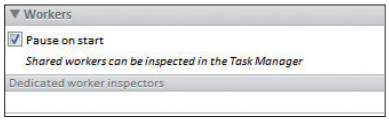


figura 24 Seguimiento de los WebWorkers en Chrome.

Workers), también están soportados como una opción dentro del menú "*Scripts*", donde hay opciones para configurar su seguimiento con cierto nivel de detalle como se aprecia en la figura 24.

Opera

Opera es uno de los fabricantes de navegadores que de forma más temprana se ha implicado en la incorporación de diversos fragmentos del estándar. Como era de esperar, también dispone de una excelente herramienta para desarrolladores, denominada **Opera DragonFly**, con opciones muy similares (e incuso superiores a las de Chrome en algunos aspectos), amén de contar con versiones localizadas para todos los idiomas en los que se instala el navegador.

Lo primero que sorprende es la cantidad de opciones configurables por el usuario, que permiten llegar a un nivel idóneo de ajuste (figura 25).

Aparte de esto, y de las opciones típicas vistas para otras herramientas, dispone de

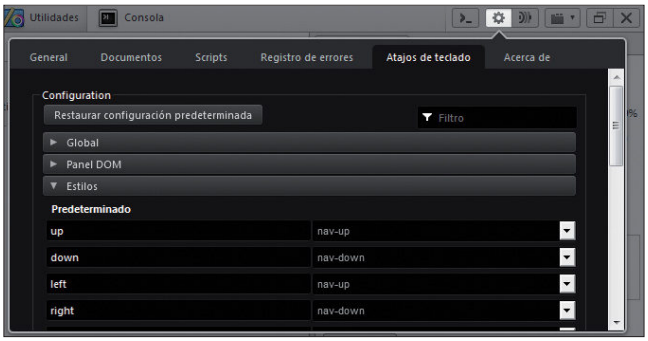


figura 25 Panel de configuración de la herramienta de desarrollo DragonFly, de Opera.

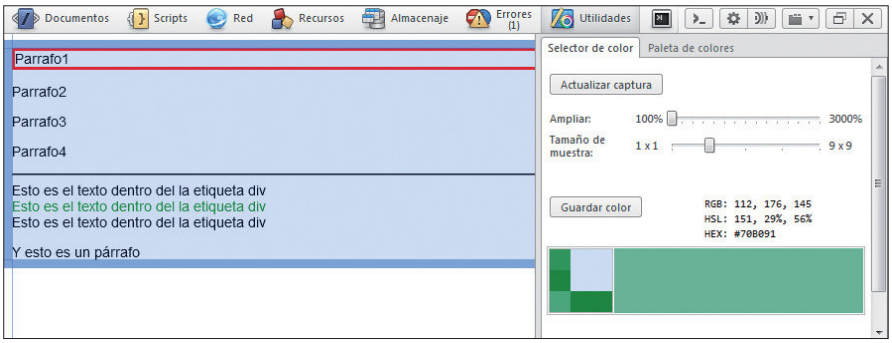


figura 26 DragonFly, mostrando una captura y detalles específicos de la misma.

algunas peculiaridades que lo hacen especialmente atractivo. Una de ellas, es la posibilidad de realizar capturas del estado de una página en un instante dado (*snapshots*), para analizar después detalladamente la evolución de la página en el tiempo, como si se tratase de instantáneas, una de las cuales se ve en la figura 26.

Otra de las pantallas que llama la atención por el nivel de detalle del contenido es la de análisis del tráfico de red, donde cada recurso (global o parcial), presenta en una ventana de contexto con todo el histórico de carga requerido para funcionar. La figura 27 muestra esta ventana en funcionamiento.

También dispone de interesantes opciones de depuración remota, análisis de código

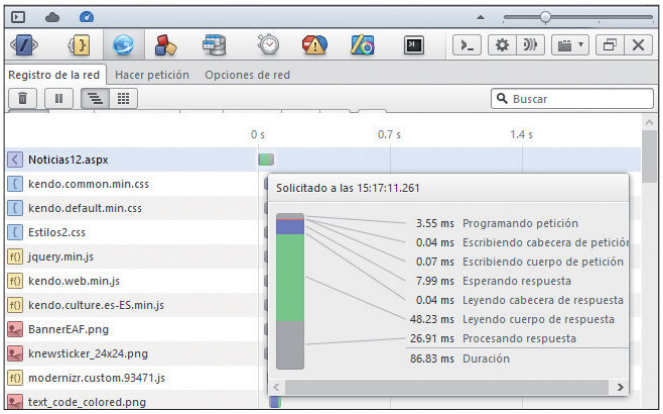


figura 27 Ventana de análisis de tráfico de red en Opera DragonFly.

dinámico, análisis de *scripts*, inspector de estilos, registro de errores, búsqueda avanzada, medidor de píxeles, registro de cambios y muchas más.

Como valor añadido, está disponible una herramienta de depuración para dispositivos móviles llamada **Opera Mobile Emulator**, y otro par de complementos (**OperaDriver** y **OperaWatir**), con los que podemos comprobar y testar cómo sería el uso

de nuestra aplicación por parte de los usuarios, incluyendo las últimas tecnologías (éstas últimas herramientas, basadas en Java).

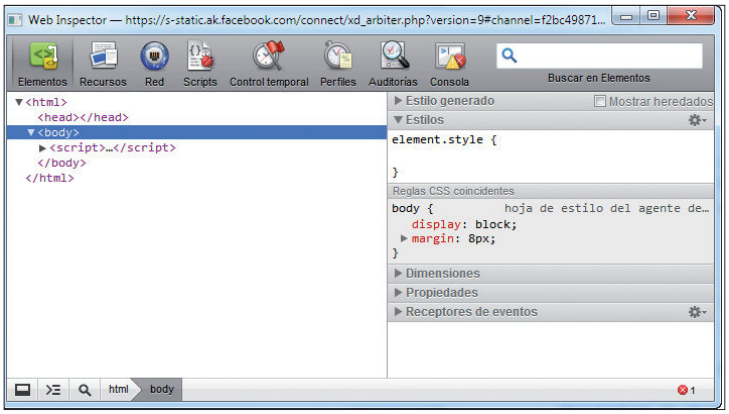


figura 28 Safari Web Inspector, mostrando las opciones generales.

Safari

Como es lógico, Safari para Windows, no dispone del conjunto funcional y las novedades que presenta para sistemas Apple, como sucede en el nuevo Safari 6, lanzado hace poco para MacBook Air. De todas formas, la versión 5.1 permite a los desarrolladores analizar su código según se ejecuta en el navegador, y utiliza ventanas de depuración y análisis similares a las de Chrome, que se lanzan en una ventana separada, como vemos en la figura adjunta:

De hecho, en las pruebas de soporte que hemos hecho, de momento es el que menor nivel de soporte del estándar ofrece, si bien suponemos que esta situación se irá corrigiendo con la aparición de sucesivas versiones.

Según sus propias publicaciones, la versión Safari 6, presenta muchos avances en el soporte del estándar y eso es lo que anuncia su sitio web como una de sus características principales.

El soporte de HTML 5 en Visual Studio

Naturalmente, las herramientas anteriores (apropiadas para la depuración, análisis y generación de informes de uso, sobre todo), no tienen por objetivo la construcción de sitios o de aplicaciones web, ni de la versión anterior, ni de la nueva.

De forma que, dada la temprana apuesta por el estándar que hizo Microsoft, como

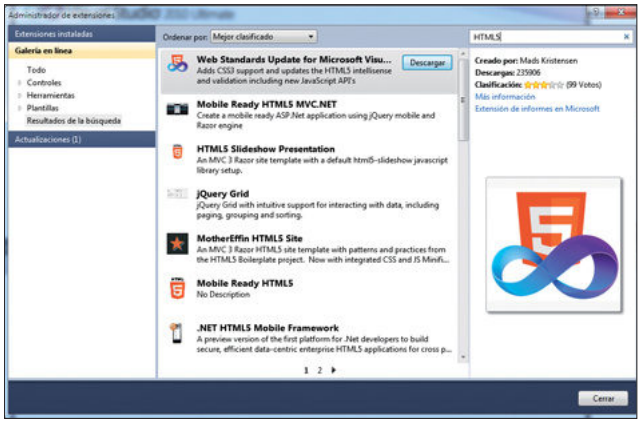


figura 29 El administrador de Extensiones de Visual Studio mostrando algunas de las extensiones disponibles, una vez filtradas.

herramienta de desarrollo, pocos pueden competir con Visual Studio. Ya en la versión 2010 encontramos un amplio nivel de soporte del estándar, que podemos mejorar notablemente con complementos (la gran mayoría gratuitos), accesibles desde la propia herramienta, en el Administrador de extensiones.

Estos complementos se añaden a la funcionalidad que ofrece el IDE, y permiten que el desarrollador que no tenga la oportunidad de contar con alguna versión de Visual Studio 2012, pueda, sin embargo, utilizar el estándar sin problemas, probar sus novedades y depurar las API relacionadas, gracias a ese soporte añadido. La figura 29 muestra la ventana de complementos, junto a algunos de los disponibles en relación con la marca HTML y similares.

La ventaja es que, en su mayoría, los propios creadores de estos complementos los mantienen actualizados con las novedades que van siendo aprobadas por la W3C, al tiempo que actualizan correcciones de *bugs* y otras opciones, facilitando así el uso de nuestra herramienta principal en todo el ciclo de vida de un sitio o aplicación Web programada con HTML 5.

El nuevo soporte de HTML 5 en Visual Studio 2012

No obstante, como Windows 8 propone un nuevo modelo de aplicaciones, ese soporte ha pasado a ser nativo en la versión 2012 de Visual Studio, por lo que recomendamos dar ese salto con la garantía de funcionamiento de todo el código actual, pudiendo aprovechar las muchas ventajas para construir sitios o aplicaciones nuevos.

Visual Studio 2012 propone una interfaz simplificada, pero más rica en posibilidades que nunca, y asume desde el primer momento que el código HTML de cualquier sitio o aplicación, se editará utilizando HTML5 (o XHTML5).

Esto es válido, no solo para las aplicaciones Web Forms, sino también para MVC3/MVC4, usando el motor Razor, e incluso a la hora de añadir una página independiente HTML a una solución cualquiera. Y si queremos mantener código antiguo sin modificar, solo tendremos que seleccionar la opción de **DOCTYPE**, y escoger el esquema W3C que más nos convenga.

El soporte del nuevo estándar en V. Studio 2012 podemos dividirlo en las 3 categorías básicas que definen la "marca" HTML5: soporte del lenguaje de marcas, soporte de hojas de estilo y soporte de API y JavaScript.

Soporte de marcado

Desde el inicio, cualquier proyecto Web o página creada con el nuevo Visual

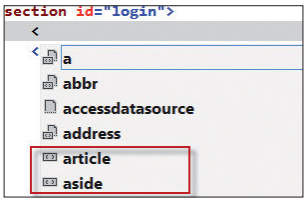


figura 30 Soporte del nuevo marcado en el editor de HTML de Visual Studio 2012.

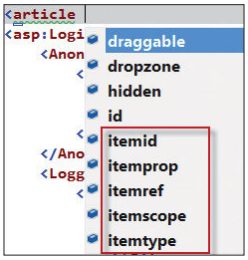


figura 31 Soporte de atributos HTML 5 en Visual Studio 2012.

Studio, asume el formato del nuevo HTML 5.Tanto si creamos una aplicación clásica Web Forms, como si optamos por un modelo MVC, el marcado que se genera desde las plantillas de proyecto, asume el prototipo de páginas HTML 5, e incluso establece una estructura inicial en la que se usan de forma directa algunas de ellas.

Independientemente, todas las nuevas etiquetas de marcado están soportadas por los editores de código, al igual que los nuevos atributos definidos en ellas, y lo mismo sucede con otros atributos nuevos de etiquetas antiguas, o marcadas como obsoletas, según se muestra en las figuras 30 y 31.

Otra opción muy solicitada por los

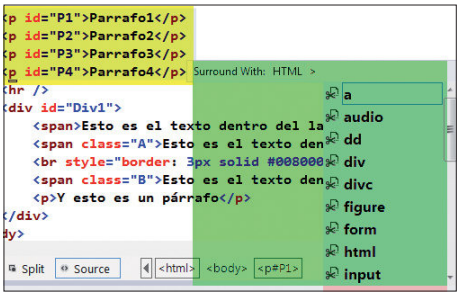


figura 32 Opciones de refactorización con Visual Studio 2012.

usuarios era la inclusión de capacidades de refactorización dentro del código HTML. Eso nos permite marcar un bloque de elementos HTML y optar por cualquiera de las opciones organizadas por categorías que nos ofrece *Refactoring*, como "Envolver con" o "Insertar Snippet", para completar el código fuente.

Por ejemplo, si queremos incluir un bloque de párrafos (elementos **<p>**) dentro de un elemento ** **, podemos marcar todos los párrafos y seleccionar "Envolver con… " ("Sorround With"), seguido de HTML y la etiqueta que sirve de contenedor, como vemos en la figura 32:

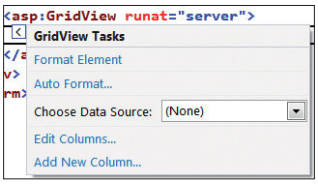


figura 33 Etiquetas inteligentes en el Editor de código HTML de V. Studio 2012.

Otra opción interesante consiste en la presencia de etiquetas inteligentes al lado de las etiquetas HTML, tanto en el marcado HTML5, como en ASP.NET. En ellas, se ofrecen opciones que tienen que ver con la programación concreta que está disponible para ese elemento.

Así que, dependiendo de la etiqueta, el menú contextual cambiará, aunque siempre ofrecerá —al menos— la opción de formatear el código del elemento y sus descendientes. La figura 33 muestra esta opción operando sobre un control **<asp:GridView>**.

Además, el soporte de marcado, se sincroniza periódicamente con los avances publicados por la W3C y las nuevas etiquetas son ofrecidas por el IDE directamente desde la Caja de herramientas o con soporte directo del editor de código.

Page Inspector

Disponemos también de una herramienta muy útil de cara a la depuración, integrada dentro del IDE: **Page Inspector**. Es una herramienta de análisis dinámico, pero

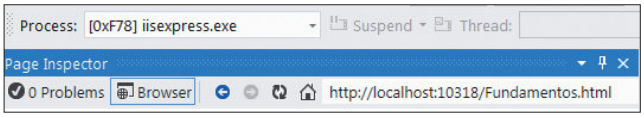


figura 34 El nuevo Page Inspector de Visual Studio 2012, mostrando el proceso activo.

integrada con los editores de código, que nos permite comprobar la ubicación, maquetación, estilos y comportamiento de cada elemento de marcado desde una ventana empotrada en Visual Studio.

Podríamos decir que dispone de buena parte de lo que existe en las herramientas de los navegadores que hemos visto antes, más la capacidad de controlar y editar el código fuente en ejecución hasta el mínimo detalle.

Y no solo sirve para analizar nuestro código, sino también cualquier página publicada, ya que admite la introducción de cualquier URL. De hecho, podemos seleccionar cualquiera de las hebras de ejecución y cambiar de una vista a otra en cualquier momento:

Además, cuenta con ventanas que habilitan la selección del tipo de inspección, y

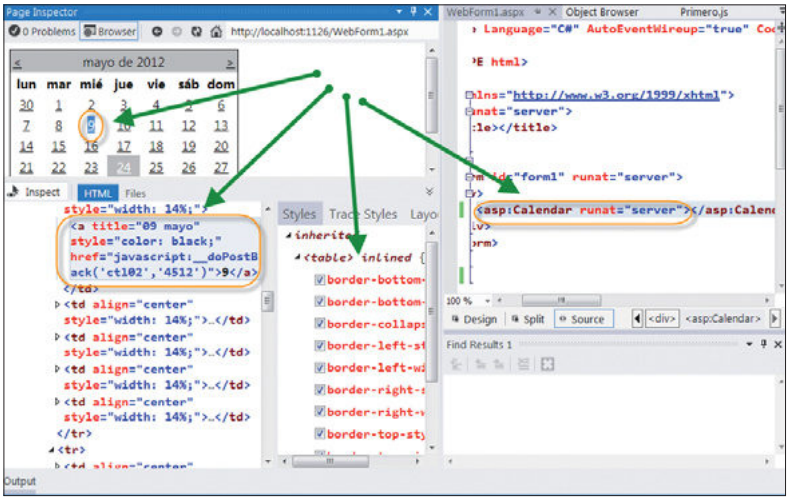


figura 35 Page Inspector mostrando las ventanas de seguimiento de CSS, código original y código final.

con un buscador de correspondencias CSS, además de otra herramienta de seguimiento de estilos (la ventana "Trace Styles").

Al mismo tiempo, realiza un doble seguimiento del código: por un lado, en la ventana del código original, de nuestro código fuente, y por otro, en la ventana de código generado, del código resultante interpretado por el agente de usuario. A esto hay que añadir la posibilidad de ver cómo se mostraría la página en cualquiera de los navegadores

que tengamos instalados en la máquina local. Una opción que se nos antoja que será de las más utilizadas por los desarrolladores web en esta versión. Podemos verlo en funcionamiento en la figura 35.

De hecho, ahora podemos ejecutar la aplicación web directamente en Page Inspector, en lugar de seleccionar un navegador concreto.

Con esta doble opción de gestión de código (generado y original), si marcamos cualquier elemento para inspección, nos mostrará —en el código generado— cuál es el elemento correspondiente y sus valores en ese instante, pero, simultáneamente, *tenemos la posibilidad de ver cuál es el ítem del código original con el que se corresponde*.

Una opción que no está presente en ninguno de los depuradores anteriores, lógicamente.

Soporte de Hojas de Estilo

Otro de los apartados interesantes (y, además, imprescindible) es el del soporte renovado de Hojas de estilo. Se trata de un soporte múltiple, que aparece en escenarios



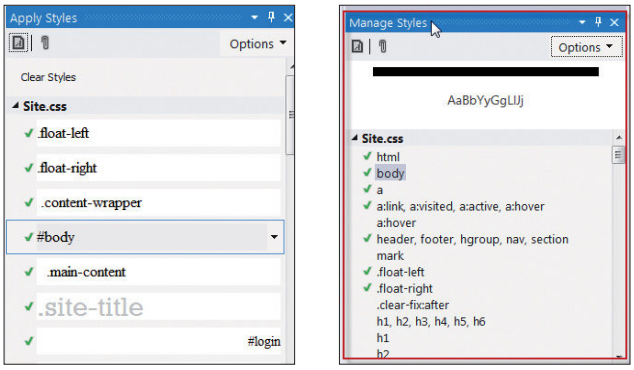
figura 36 Nuevas ventanas asociadas a Hojas de Estilo en Visual Studio 2012.

distintos del proceso de desarrollo.

Por ejemplo, ahora es posible cancelar o modificar la aplicación de estilos para ver el resultado potencial en la ventana de ejecución de Page Inspector, y modificar atributos o eliminar temporalmente cualquier elemento y ver el resultado igualmente. De hecho, el soporte dinámico de Page Inspector permite hacer un seguimiento de qué reglas de estilo son aplicables a un elemento en un momento dado de la ejecución, y filtrar cualquier efecto hasta averiguar la causa de su aspecto visual.

Además, se observa también la presencia de 3 nuevas ventanas relacionadas con CSS dentro del menú "VIEW", que permiten, respectivamente, comprobar, administrar y aplicar de forma global o individual elementos de estilo a etiquetas del código de marcas:

Las figuras 37, 38 y 39 presentan las ventanas de edición que permiten el desglose de todo lo relacionado con estilos con gran nivel de detalle. En "Apply Styles", se observa de forma dinámica como se aplica un estilo a un elemento, y se puede deshabilitar, o eli-



figuras 37 y 38 Ventanas de edición CSS "Apply Styles" y "Manage Styles"

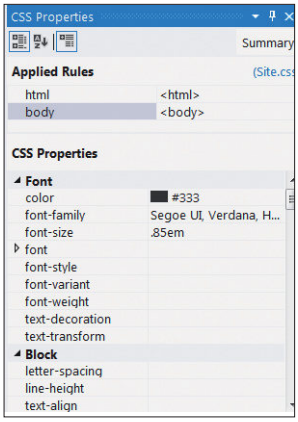


figura 39 Ventana "CSS Properties".

minar todos los estilos en el DOM para apreciar mejor la estructura del documento. En el apartado "Manage Styles", podemos comprobar todo lo relacionado con una hoja de estilos concreta y ver igualmente el resultado visual sobre el ítem seleccionado.

Finalmente, en la ventana "CSS Properties", se analiza, elemento por elemento, cuales son los estilos aplicados sobre él, incluyendo, claro está, aquellos que han quedado desactivados por una instrucción de ámbito más local, como aparece en la figura 39.

Además de lo anterior, cuando estamos editando un fichero de hojas de estilo, el propio Editor empotra ventanas complementarias para ayudar en el proceso de edición, tal y como vemos en la figura 40, donde se muestran las opciones para la selección de

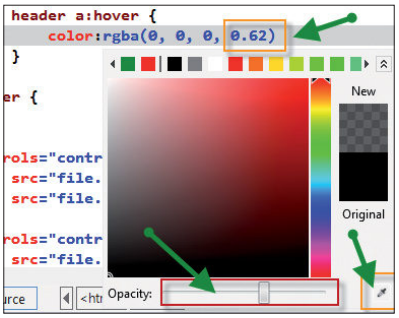


figura 40 Editor de colores CSS en Visual Studio 2012.

reglas de color.

Hay que notar que la ventana no necesita de un botón "Aceptar", ya que se realiza un análisis de código dinámico todo el tiempo, y también dispone de un canal *alfa* dentro

```
<style>
    header {
        color: #f00;
    }

        header a {
            color: #008000;
        }

            header a:hover {
            }
</style>
```

Listado 1: Indentación en estilos CSS

de la barra de deslizamiento "*Opacity*", de tal forma que si la utilizamos, modifica la definición del color con su valor correspondiente en el código fuente de la regla, para recoger esta definición en el código, sin necesidad de intervención del usuario.

De igual manera, en la parte inferior derecha observamos la presencia de un *Eye Dropper* o selector dinámico de color que nos permite desplazarnos a cualquier otra zona o ventana (incluso fuera de Visual Studio) y seleccionar el color deseado, obteniendo una captura exacta del color marcado.

Otra característica nueva es la indentación jerárquica de elementos relacionados, de forma que, resulte más fácil seguir el orden en que se aplican los estilos a distintos elementos. Podemos ver este comportamiento automático en el código siguiente:

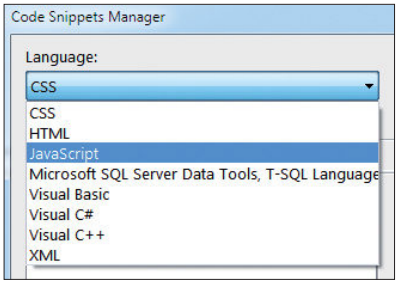


figura 41 Snippet Manager mostrando las opciones de selección de código.

También hay opciones muy populares de la edición de código .NET que han pasado a formar parte del editor de CSS 3, como es el caso de los *snippets* de código. Recordemos que, en la versión 2010, solo existían unos pocos *snippets* vinculados con HTML, pero no con CSS.

Ahora contamos con *snippets* separados para los 3 tipos de código del estándar,

```
h1 {
      -ms-transform: rotate(-90deg);
      -moz-transform: rotate(-90deg);
      -o-transform: rotate(-90deg);
      -webkit-transform: rotate(-90deg);
      transform: rotate(-90deg);
   }

@font-face {
            font-family: 'font name';
            src: url('/content/file.eot');
            src: local('☺'), url('/content/file.woff')
            format('woff'), url('/content/file.ttf') format('truetype');
          }
```

Listado 2: código generado por las macros *transform* y *@font-face*

como vemos en la figura adjunta:

Las opciones que ya existían (HTML y JavaScript), han incrementado muchísimo la oferta de *snippets* disponible, y en la opción CSS, contamos con varios modelos, más la opción de crear nuestras propias entradas.

La forma de acceso es tipo "macro" —pulsando el tabulador—, y encontramos varias que nos dan la opción adecuada para el código CSS que entienden todos los na-

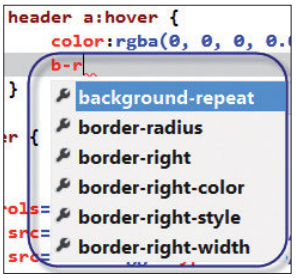


figura 42 Intellisense del Editor de CSS3 mostrando selección por abreviaturas

vegadores. Por ejemplo, la macros `transform` y `font-face`, nos generan, respectivamente, el siguiente código:

Donde `-ms-`, `-moz-`, `-o-` y `-webkit-` se corresponden respectivamente con las extensiones de IE, FireFox, Opera y Chrome/Safari, que comparten la misma definición.

De hecho, el editor incluso admite abreviaturas de las iniciales de las reglas para seleccionar dinámicamente las que buscamos. Por ejemplo, si tecleamos las iniciales de "border-radius" (***b-r***), todos los elementos compuestos que incluyen esas iniciales aparecen automáticamente, como se ve en la figura 42.

Como nota complementaria, recuerde el lector que la documentación oficial de la W3C, así como de la propia de Microsoft y otros fabricantes, recomiendan que se realice la comprobación del soporte "por característica" y no por versión del navegador, suministrando una alternativa o *fallback*, para cuando una de ellas no esté presente.

Soporte de JavaScript

Finalmente, el otro apartado donde contamos con un soporte renovado y muy mejorado respecto a versiones anteriores es JavaScript. En el capítulo anterior, ya comentamos que el analizador se construyó completamente desde cero, basándose en la implementación de Chakra y ofrece una gran rapidez de ejecución.

En primer lugar, y por primera vez en todas las ediciones de Visual Studio, el soporte del estándar ECMAScript es total. Además, el mecanismo extendido de edición de código (Intellisense) reconoce ahora los caracteres introducidos por el usuario, ofreciendo en el menú contextual solamente aquellas palabras que contienen el texto introducido hasta ese momento, dependiendo de dónde nos encontremos (es sensible al contexto). Por ejemplo, si en la instrucción anterior se carga un fichero JavaScript, las funciones definidas allí, estarán automáticamente disponibles desde la instrucción si-

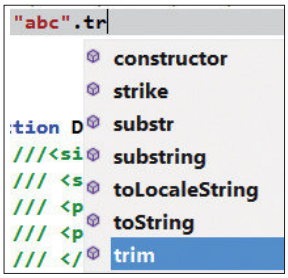


figura 43 El editor de JavaScript mostrando opciones avanzadas

guiente.

También se han incorporado características de verificación de código (*syntax checking*) ya presentes en otros lenguajes. En la figura 43 podemos ver dos de estas novedades: por un lado, el soporte de la función `trim()` definida en el nuevo ECMAScript, y por otro, la lista reducida de valores posibles que ofrece el editor, filtrando las palabras que contienen las dos primeras letras de "**trim**".

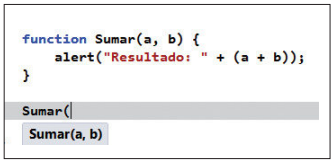


figura 44 Intellisense asociado a JavaScript.

En el menú contextual del editor, se ha incluido, además, la opción "Ir a Definición", de forma que podemos desplazarnos fácilmente a cualquier definición de otra función JavaScript, dondequiera que esté ubicada. Incluso si se encuentra en un fichero distinto.

De esta forma, integramos la búsqueda local de código con la que aportan las librerías de programación general del lenguaje. Veremos esto más adelante con algo más de detalle, en el apartado "Script Loader".

Una muestra del reconocimiento de la signatura de funciones aparece en la figura 44:

La opción de identificar 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, ahora también podemos indicarle al editor que mantenga referencias implícitas sobre librerías, que serán tenidas en cuenta durante todo el proceso de edición. Ya contiene varias referencias predeterminadas, y podemos

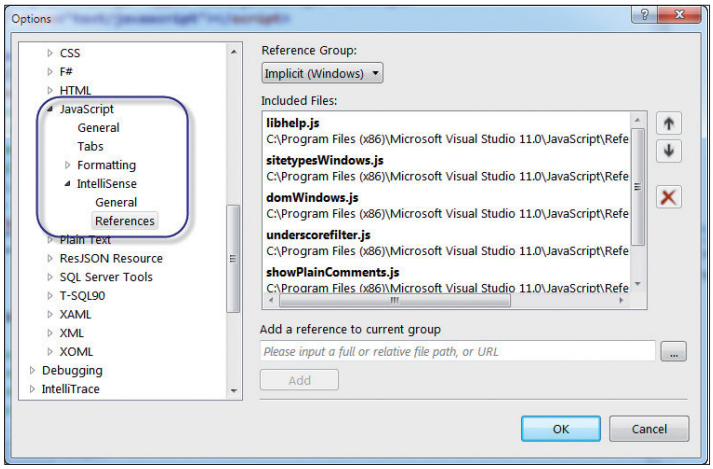


figura 45 Configuración de referencias externas para el editor de JavaScript.

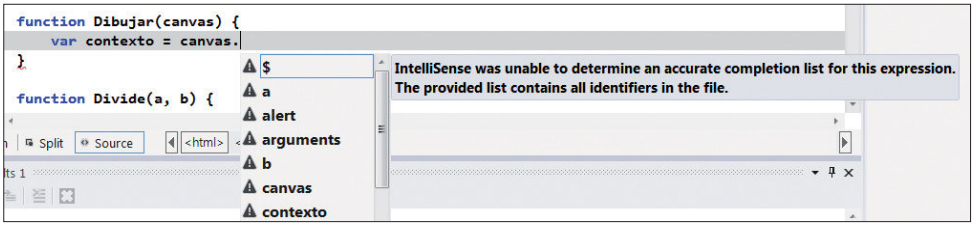


figura 46 Reconocimiento de identificadores definidos.

incluir las que necesitemos, como muestra la figura 45.

Otra opción importante relacionada con la edición es el comportamiento que permite mostrar los identificadores definidos por el usuario, y detectar cuáles tienen valores indefinidos (*undefined*). Podemos ver esta opción en la figura 46, donde vemos la lista de identificadores definidos en el fichero activo, junto al aviso de que el argumento que supuestamente recibe la función (*canvas*), no es reconocido por el editor.

En el caso de que necesitemos reconstruir los datos de Intellisense, ahora disponemos del comando [CTRL]+[SHIT]+[J] para hacerlo automáticamente. Y hay muchas otras opciones que se han añadido para mejorar el soporte presente en la versión 2010, como la precisión a la hora de mostrar información en las funciones con nombre.

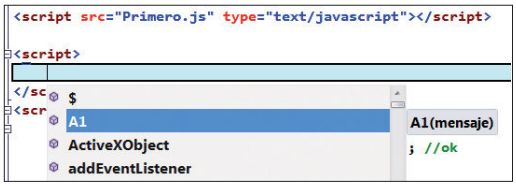


figura 47 El *"Script Loader"* del editor de JavaScript en acción.

Script Loader

Este mecanismo añade dinámicamente referencias del contenido de otro archivo JavaScript al que apunte una línea de código que lo cargue dinámicamente.

En la figura 47 podemos ver cómo el fichero **`Primero.js`** está referenciado en la línea anterior, y la función definida en él —*A1(mensaje)*— aparece correctamente indicada a partir de la siguiente línea de *script*:

nota

Los usuarios de *snippets* de código de la versión 2010 deberán saber que tienen que cambiar el atributo Language de la definición para que indique *JavaScript* en lugar de *JScript*.

Como una parte añadida de estas capacidades, ahora también se puede depurar inmediatamente en el editor un archivo descargado de una fuente externa al programa. En realidad, cualquier archivo JavaScript externo.

Y a todo este paquete de mejoras hay que añadirle muchas de las que existían para otros lenguajes: resaltado de palabras, control y resaltado de las llaves de apertura/cierre, validación de expresiones regulares, codificación de colores, soporte de la directiva **`use strict`** presente en ECMAScript 5, explorador del DOM, consola JavaScript y como apuntábamos antes, el apartado de *snippets* de código para este lenguaje.

En el capítulo 6, revisaremos el lenguaje JavaScript 5 como parte del recorrido por las API de HTML5 y sus nuevas características.

Microsoft Expression Blend para Visual Studio 2012

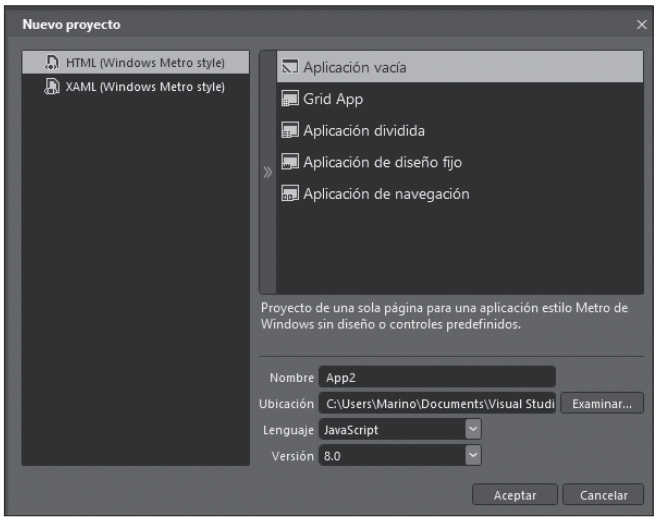


figura 48 Ventana de selección del modelo de aplicación en Blend para Visual Studio 2012.

Con la aparición de Visual Studio 2012, también se instala una versión de la herramienta Expression Blend, especialmente pensada para potenciar la parte del diseño de las aplicaciones que utilizan XAML o HTML en la interfaz de usuario.

Aunque la versión de que disponemos en el momento de escribir esto no es la versión final, sino la Release Candidate, gran parte de la funcionalidad ya está presente aquí.

Existe una consideración a tener en cuenta —al menos de momento— respecto a su uso con el estándar, y es que esta versión solo permite la creación de aplicaciones Windows 8, aunque los lenguajes utilizados sean los mismos.

Plantillas disponibles

Blend, incluye por defecto varias plantillas para los dos modelos de este tipo de aplicación editables con Blend: la que usa XAML + C# ó VB.NET, y la que utiliza HTML 5 y CSS 3 con JavaScript. Si optamos por ésta última, ofrece un paquete de o p -

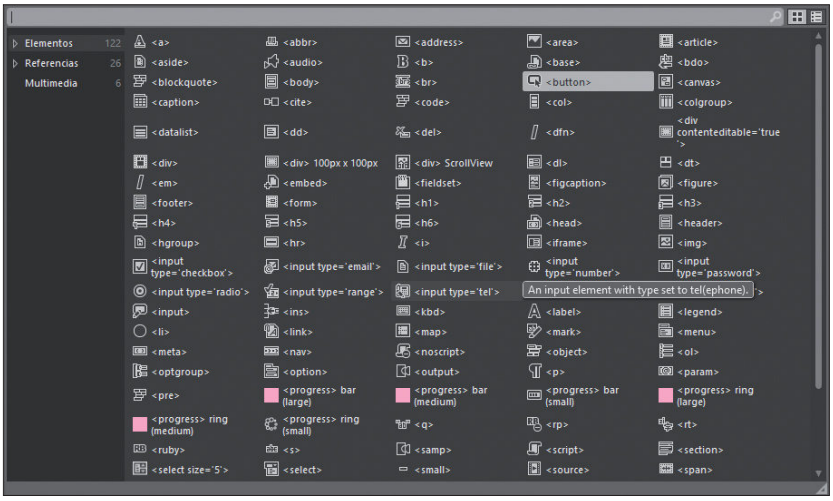


figura 49 Cuadro de Herramientas mostrando los "Activos" de Blend.

ciones que aparecen en la figura 48.

Una vez que hemos optado por una de ellas, el entorno dispone de soporte completo, tanto en diseño como en edición de código, y permite —elemento por elemento— todas las opciones definidas por el estándar, adaptando, eso sí, la metáfora de desarrollo habitual de .NET a este nuevo contexto.

Por ejemplo, en el área que para XAML estaba dedicada a los controles, aquí tenemos todos los elementos definidos en HTML5, más un conjunto de controles creados para su uso con aplicaciones Windows 8, que de esa manera "redefine" las interfaces de usuario de estas aplicaciones en ideas visuales bien conocidas. Esta opción podemos verla en la figura 49.

Todos los elementos de HTML (nuevos y antiguos, más sus atributos, más todas las nuevas opciones de CSS3 y JavaScript) están recogidas en este apartado, si bien los controles creados para aplicaciones Windows 8 solo puede usarse para estas aplicaciones, ya que se basan en la librería **Windows Library for JavaScript**, que podemos vemos en el pro-

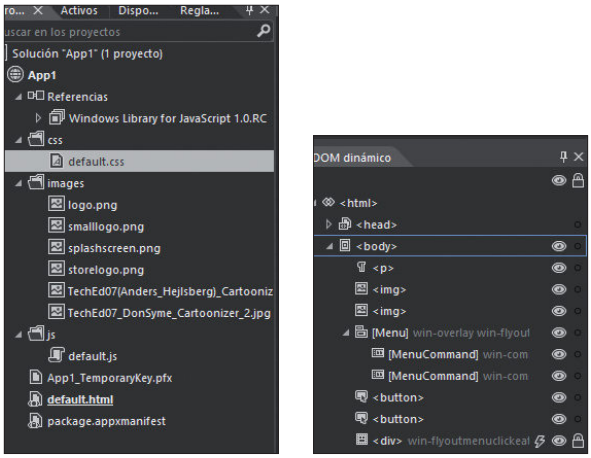


figura 50 Explorador de soluciones y Ventana "DOM Dinámico".

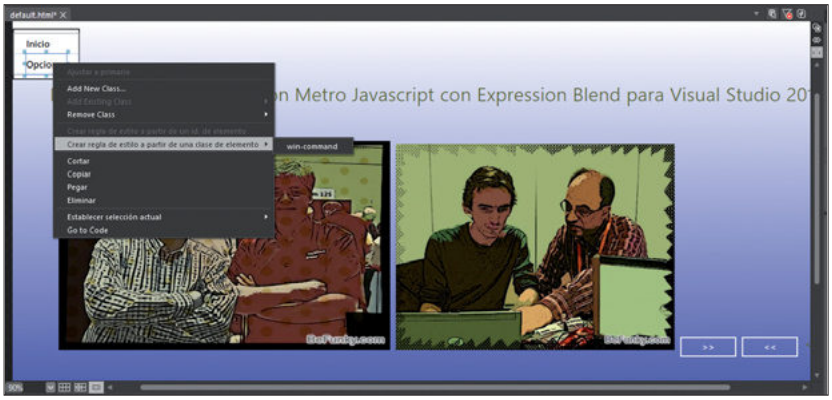


figura 51 El editor visual de Blend.

yecto dentro del apartado "Referencias".

Por lo demás, tenemos pleno acceso a todos los recursos de la aplicación y la capacidad de añadir otros o modificar los existentes. El Explorador de Soluciones nos mostrará la información detallada de cada componente de la solución al estilo de las versiones anteriores, y también contamos con una ventana denominada "DOM Dinámico", donde podemos ver el árbol generado por los elementos HTML y aplicar a cada

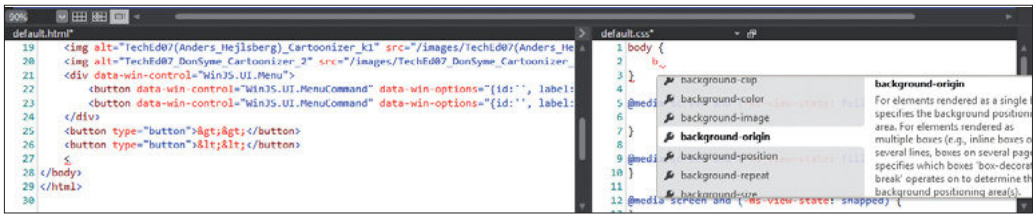


figura 52 Ventanas de edición de código (HTML y CSS) en Blend.

uno de ellos las acciones de diseño que permite Blend, como ocultarlo en la ventana del diseñador o bloquear cualquier posible modificación sobre él.

Nos podemos dar una idea de esto en la figura 50, que muestra el Explorador de soluciones y la ventana del DOM Dinámico en funcionamiento.

Y, como siempre, en el centro de la pantalla, se sitúa el editor visual, que funciona de forma prácticamente idéntica a la de XAML, pero adaptando su conjunto de mensajes, menús contextuales, etc., a este modelo, como se ve en la captura correspondiente a la

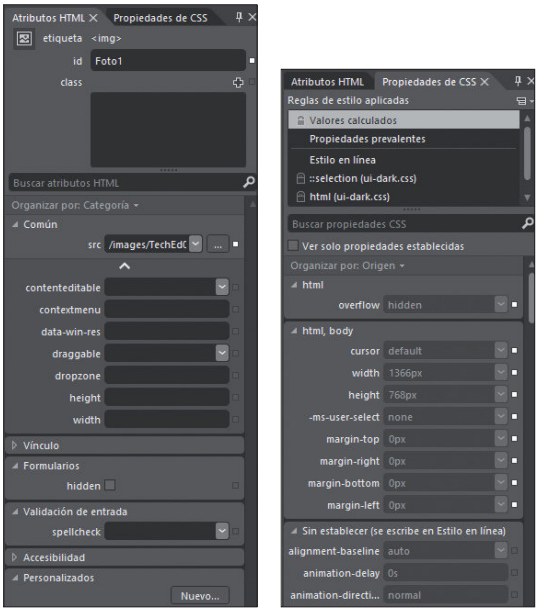


figura 53 Ventanas de propiedades HTML y CSS respectivamente, mostrando los valores aplicados al elemento seleccionado en el editor visual.

figura 51.

En la parte inferior (y predeterminada), se ubican ahora dos sub-ventanas de código que muestran de forma simultánea el marcado HTML 5 y la hoja de estilo enlazada que se esté aplicando a un elemento activo (si lo hay).

En ambos casos, disponemos de Intellisense, como era de esperar y puede verse en la figura 52.

Y, por último, y —aunque esta no es toda la oferta completa— concluimos este repaso inicial de opciones con dos solapas alternativas que ocupan una ventana que sería la equivalente en Visual Studio a la ventana de Propiedades. Solo que, en este caso, esas propiedades se expresan en dos lenguajes diferentes: por una parte, los valores de los atributos HTML, y por otra, los de las reglas CSS 3 aplicables al elemento.

Muchos de estos atributos se muestran en ventanas propias, incluyendo las extensiones que nopertenecen al estándar y que los fabricantes utilizan mientras terminan la implementación o concluye la especificación oficial. La figura 53 muestra esta característica.

También se han añadido muchos pequeños detalles útiles en el proceso de edición de estilos, como indicadores de los valores de la caja de un elemento (`margin` y `padding`), ayudas para el escalado visual, indicadores de cuando un elemento sufre una transformación, edición múltiple, menús contextuales para la creación de estilos, añadir y/quitar estilos, vistas por código fuente alternativas, indicación de selectores inválidos con Intellisense, soporte de gradientes y transformaciones, y un largo etcétera.

En lo que a JavaScript se refiere, el editor de código —de momento— no muestra las características de Intellisense, sino solo el coloreado de las palabras reservadas. Probablemente, esto esté presente en la versión final.

En cualquier caso, las modificaciones y novedades de esta versión de Blend se aprecian inmediatamente en el editor visual, tanto si es en HTML, como CSS. Y esto incluye las nuevas opciones de animación propias de CSS3 (nos referimos a transformaciones y animaciones de diversos tipos en las que es establece una duración, normalmente indicada en milisegundos), al estilo del soporte que encontrábamos para Silverlight o WPF.

En suma, Blend para Visual Studio 2012 resulta el complemento perfecto para los diseñadores Web o para desarrolladores que quieran una herramienta avanzada y un soporte completo del estándar HTML 5.

Referencias

IE Blog. Sobre el desarrollo con Internet Explorer: http://blogs.msdn.com/b/ie/

MSDN Microsoft, "Guía de Windows Internet Explorer 10 Release Preview para desarrolladores", `http://msdn.microsoft.com/library/hh673549.aspx`

Justin Garret, "*Get 30% better site performance in IE10 with HTML5*", `http://windowsteamblog.com/ie/b/ie/archive/2012/06/14/get-30-better-site-performance-in-ie10-with-html5.aspx`

Vishal Gupta, "*How to Use All New Internet Explorer Platform Preview Build Features in IE Public Beta Build?*", `http://www.askvg.com/how-to-use-all-new-ie9-pp6-features-in-recently-released-ie9-public-beta-build/`

Fiddler, sitio oficial: `http://fiddler2.com/fiddler2/`

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`

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)`

BlendInsider, "The Nitty Gritty—Detailed List of what's new in Blend for Visual Studio 2012 RC", `http://blendinsider.com/technical/the-nitty-gritty-detailed-list-of-whats-new-in-blend-for-visual-studio-2012-rc-2012-05-31/`

capítulo 3

HTML 5: nuevas etiquetas

Así pues, ¿qué debemos entender por HTML 5? No existe una definición oficial, aunque sí unanimidad al referirse a él como el nuevo estándar de la Web. De todas formas, puede leerse una definición establecida por la W3C, con la ayuda de otras entidades colaboradoras que llevan a cabo la elaboración de otras partes, tales como las API que hemos citado en el primer capítulo.

Lo que el borrador actual del estándar indica claramente son los siguientes puntos de trabajo:

1. Definir un único lenguaje llamado HTML 5, que pueda 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
4. Introduce nuevas etiquetas y API para tecnologías emergentes, tales como las aplicaciones web.

El prablema de la WHATWG

En el verano de 2012 se han producido noticias algo desestabilizadoras para el estado del estándar, que se relacionan con la otra entidad implicada en la construcción: la **WHATWG** (o *Web Hypertext Application Technology Working Group*), 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, recientemente, y coincidiendo con el paso de Hickson a Google, éste ha declarado[1] que "su objetivo es 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), "para poder 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 duro 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 sería de desear que los navegadores unificasen sus criterios de acuerdo con unos mismos principios básicos.

El marca de trabaja y las abjetivas

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 "*c*ualquier 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 tenían que reflejarse en el modelo. Y todo esto, teniendo en cuenta que no se podía romper la compatibilidad con los muchos millones de Webs existentes, por lo que, aunque algunas etiquetas se declaran ahora como obsoletas, se asumen con el estado de *conforming* (o sea, todavía utilizables).

[1] *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

Campatibilidad 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 cuya funcionalidad pertenece 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.

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 e´ 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.

Sintaxis general

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. 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**.

Así pues, la primera instrucción de todo documento HTML 5, debe indicar, en una u otra sintaxis, su tipo *mime*, y esto se realiza en HTML mediante la directiva `<!doctype>`. Lo que, en HTML 5, se reduce al código 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 pasa a ser simplemente la primera línea del código anterior, sin referencias a ninguna página oficial.

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 prefiramos usar la 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 tipa de dacumenta: DOCTYPE

Detrás **DOCTYPE** hay una larga historia. La definición **DOCTYPE** ha establecido habitualmente el tipo de documento del texto que seguía a continuación y esto es algo que no estaba previsto en las especificaciones. Pero es que 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 existente hasta ese momento se basaba en las peculiaridades de los exploradores dominantes entonces: Internet Explorer 4 y Netscape 4. Más adelante, y especialmente IE5 (en su versión para MAC) resultó 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 es debido a que, en versiones anteriores, **DOCTYPE** se basaba en el estándar SGML, y requería de la presencia de una DTD (*Definición de Tipo de Documento*). Ahora, los navegadores soportan directamente esta sintaxis.

Cadificación de caracteres (Encading)

Finalmente, conviene recordar 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, y en HTML 5 se dispone de 3 formas de indicar la codificación[2]:

- Desde el punto de vista del transporte, usando la cabecera **HTTP Content-Type**.
- 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, ahora 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. Completaremos la descripción con ejemplos que ayuden a ubicar el alcance y el propósito de las nuevas etiquetas.

---

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

HTML5: las nuevas elementas

Bajo este apartado se recogen muchas sugerencias que los usuarios de todo el mundo habían apuntado como necesarias. Aportan novedades en la semántica, estructura, multimedia, formatos de entrada, etc. La lista completa de elementos nuevos es la que se muestra en el siguiente cuadro, junto a una breve descripción inicial:

Nuevas etiquetas en HTML 5	
section: representa un documento genérico o una sección de la aplicación. Puede ser usada junto a elementos `h1`, `h2`, `h3`, `h4`, `h5`, y `h6` para indicar la estructura del documento.	`article`**:** representa una pieza independiente del contenido de un documento, como una entrada de un blog o artículo de un periódico.
`aside`**:** representa un elemento de contenido ligeramente relacionado con el resto de la página.	`hgroup`**:** representa el encabezado de una sección. **(Nota: en un principio, se consideró abandonado, utilizar con cuidado)**[3]
`header`**:** representa un grupo de ayudas introductorias o de navegación.	`footer`**:** representa un pie de página de una sección y puede contener información sobre el autor, la información de copyright, etc.
`nav`**:** representa una sección del documento para la navegación.	`figure` **/ figcaption:** el primero, representa una pieza auto-contenida de información a la que normalmente se hace referencia como una sola unidad en el flujo principal del documento. La segunda, se puede utilizar como título (es opcional), y puede ir ubicada al inicio o al final del elemento `figure`.

[3] http://lists.w3.org/Archives/Public/public-html/2011Nov/0044.html

Nuevas etiquetas en HTML 5	
video y **audio**: Permiten la inclusión directa de contenidos multimedia. Ambos proporcionan una API para que los desarrolladores puedan definir su propia interfaz de usuario, pero también es una manera de activar una interfaz de usuario proporcionada por el agente (navegador).	**source** y **track**: Los elementos **source** se utilizan junto con los elementos video y audio si hay varias fuentes disponibles de diferentes tipos. **Track** proporciona secuencias de texto para el elemento **video**, que se pueden configurar mediante el uso de un grupo de atributos que permiten indicar información de metadatos.
embed: se utiliza para la inserción de un complemento (*plugin*).	**mark**: representa una renglón de texto en un documento, marcado o resaltado con fines de referencia, debido a su importancia en otro contexto.
progress: representa el grado de finalización de una tarea, como una descarga o cuando se realizan operaciones de larga duración.	**meter:** representa una medición, como puede ser el porcentaje de uso de un disco.
time: representa una fecha y hora.	**ruby**, **rt** y **rp** permiten marcar anotaciones *ruby*.
bdi: representa un intervalo de texto bidireccional que debe ser aislado de su entorno a efectos de formato de texto.	**wbr:** representa un lugar, dentro de una cadena muy larga sin espacios, donde puede situarse un salto de línea.
canvas: se usa para procesamiento dinámico de mapa de bits gráficos sobre la marcha, como gráficos o juegos.	**command**: representa un comando que el usuario pueda invocar.
details: representa información adicional o controles que el usuario puede obtener bajo demanda. El elemento **summary** ofrece su resumen, leyenda o título.	**datalist**: junto con el nuevo atributo **list** para el elemento **input**, puede utilizarse para hacer cuadros combinados (*ComboBoxes*).

Nuevas etiquetas en HTML 5	
keygen representa un mecanismo de control para la generación de un par de claves.	**output**: indica algún tipo de salida, como la que tiene lugar después de un cálculo mediante *scripting*

Tabla 2. Lista de los ´uevos eleme´tos e´ HTML 5

Todas estas palabras reservadas definen etiquetas nuevas (*tags*). Por supuesto, todas disponen de sus correspondientes atributos que permiten personalizar su comportamiento, y tal como sucedía con 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 todas) las etiquetas, como **style**, por ejemplo.

Cambias genéricas para tadas las elementas. Atributas glabales

Dentro del 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

Atributo	Descripción	Valores posibles
`dir`	Especifica la dirección del texto. (Derecha a izquierda, o izquierda a derecha)	• `ltr` • `rtl`
`draggable` arrastrar.	Indica si el elemento se puede	• `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…

Atributo	Descripción	Valores posibles
`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 3. Atributos globales para todos los eleme´tos de HTML 5

Advierta el lector que, en algunos casos estos atributos ya existían en la especificación anterior, pero no tenían el carácter global que tienen ahora.

Las nuevas etiquetas, par categarías

La importancia del grupo de etiquetas de la tabla 2, no es equivalente. Y por otra parte, tiene sentido dividirlas por categorías según su propósito, lo que ayuda bastante a su utilización práctica. Vamos a desglosarlas según este criterio.

Etiquetas estructurales a 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 armazón 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.

Hasta ahora, con HTML 4.01, la arquitectura de las páginas se basaba principalmente en elementos `<table>` y `<div>`. En sí, estas etiquetas no ofrecen ninguna indicación acerca del contenido, ni mucho menos 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 esta circunstancia, en las webs de hoy existe mucha información específica, que tiene en sí un significado común (artículos, fechas, comentarios, entradas, calendarios, etc.).

Este problema, puede ser paliado en parte gracias a las nuevas etiquetas **`section, article, aside, header, footer, nav`** y **`figure`**, que hemos visto en el cuadro anterior. 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.

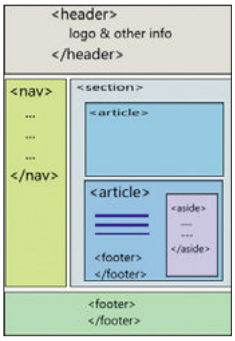


figura 1 Esquema clásico de ubicación de elementos estructurales en un blog.

La figura 1 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.

Etiqueta <Sectian>

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, asociarles 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 nume-

radas 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 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, como vemos, hay dos secciones (Artículos y Comentarios), y —a su vez— podemos dividir el primer artículo en otras dos secciones. Visual Studio 2012, nos mostraría en la ventana de Esquema de documento una estructura como la de la figura 4:

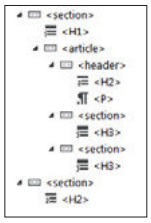


figura 2 Ventana de esquema de documento del código anterior en Visual Studio 2012. Esquema clásico de ubicación de elementos estructurales en un blog.

<sectian> y la nación de esquema de un dacumenta

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**…) genera un 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.

Expresado en términos de HTML tradicional, podríamos indicar 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>New Information</h1>
</body>
```

El esquema de documento generado sería como sigue:

1. Main Paragraph
   a. Second Level…
   b. Another Second Level

2. New 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, y la fusión de más de un documento es propensa a errores. Además, 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 con-

tenidos 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, como puede verse en la figura 4.

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

Etiqueta <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.

Etiqueta <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 completa el tema que se explica de forma que resulta una ayuda adicional 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 sirve de añadido, 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 se muestra en la figura 3:

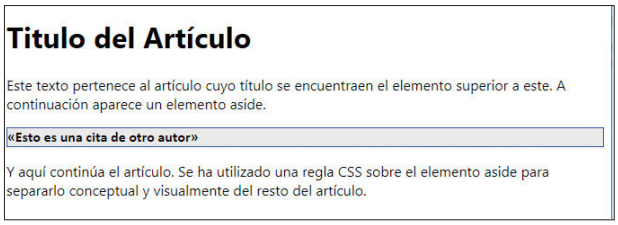


figura 3 Salida del código anterior en el que el elemento <aside> se diferencia visualmente de su contenedor, utilizando reglas CSS.

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.

Etiqueta <header>

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

Su connotación semántica se relaciona con el elemento al que sirve de cabecera, ya que puede albergar datos sobre el propio contenido (autor, fecha, lugar, fuente, etc.) o la división en partes del mismo, aportando enlaces 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.).

Etiqueta <faater>

Especialmente pensado para pies de página, pero hay que entenderlo en un sentido similar a **<header>**, esto es, como una información adicional del 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 una etiqueta diferente contenga varias etiquetas **<footer>**, si resulta adecuado.

El código siguiente es un ejemplo con un elemento **<section>** que contiene otro elemento **<article>**. Mientras éste incluye 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.

Etiqueta <nav>

Este elemento representa un fragmento de una página que enlaza con otras páginas del mismo sitio o aplicación, 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 habituales de referencias en un artículo, o los que indican páginas complementarias (*Condiciones legales*, *Copyright*, *Política de privacidad* etc.), no son candidatos a aparecer aquí, porque 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 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). Ver listado 1.

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 cambio de tamaño en la salida de dos elementos **<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).



figura 4 Salida del código anterior en Internet Explorer.

```
<!doctype html>
<html>
<head>
    <title>Prototipo de ´log: El Fénix</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="´log.html">´log</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 - www.ElAveFenix.net</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>
```

Listado 1

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 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 relacionado con ese elemento, y no un simple cambio de la información presentada. En el apartado dedicado al elemento `<command>` explicamos con más detalle el funcionamiento conjunto de estos elementos.

Implementación de una entrada de un blag

Si llevamos estas ideas a la típica entrada de un blog, en la que el bloguero escribe un artículo y se admiten respuestas y comentarios, podríamos concebir un esquema como el que aparece en la figura 5. Adviértase como 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.

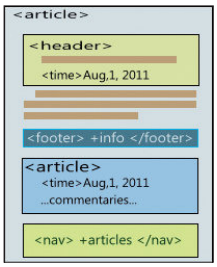


figura 5 Estructura de entradas de un blog aplicando la nueva filosofía semántica propuesta por HTML5

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 volup-
tas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione
voluptatem sequi nesciunt.
            </p>
            <footer>
                <em>El informador, Redmond, Wa.</em>
                <a href="www.theReporter.com">(The Reporter)</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 necessita-
tibus saepe eveniet ut et voluptates repudiandae sint et molestiae non recusandae aut per-
ferendis doloribus asperiores repellat...)</strong>
        </article>
    </section>
```

Generando una salida como la que vemos en la figura 6.

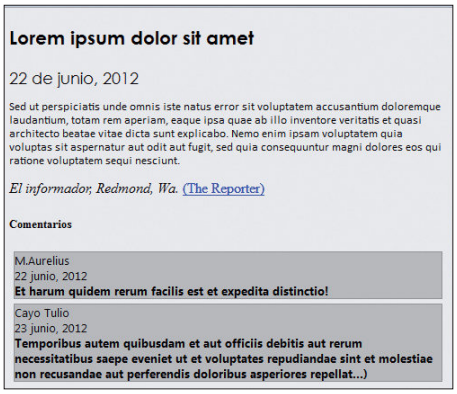


figura 6 Salida en IE9/10 del código anterior.

Elementa <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="/pictures/landscape.jpg">
    <figcaption>West Virginia</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 información especial respecto al artículo o sección a la que pertenece, generando una explicación complementaria adecuada a un escenario dado: 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="Imagenes/Hydrangeas.jpg" />
        <img src="Imagenes/Chrysanthemum.jpg" />
        <img src="Imagenes/Tulips.jpg" />
        <figcaption>Motivos florales en Windows 7</figcaption>
    </figure>
```

Lo que (añadiendo un par de ajustes de estilo, nos puede generar una salida como la de la figura 7):



figura 7 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, ajustándose magníficamente a los contextos actuales, donde es preciso programar para dispositivos diversos.

En el capítulo dedicado a CSS 3, veremos cómo se pueden combinar estas características con las nuevas reglas dinámicas de Transiciones y Animaciones, y las directivas **@media** para comprobar la superficie disponible en el dispositivo de destino y ofrecer una experiencia de usuario diferenciadora, que se ajuste siempre al contexto de visualización, sin requerir siquiera del concurso de funciones JavaScript.

Nata sabre este canjunta de etiquetas semánticas

Con estas etiquetas estructurales, estamos añadiendo vínculos entre elementos y estableciendo dependencias entre fragmentos de código de una página y por tanto, favoreciendo la implementación de la Web semántica. Quede claro que su papel no es contenedor (o estructural, si se quiere), sino relacional: definen relaciones lógicas entre los contenidos.

Eso es algo de lo que no disponíamos hasta ahora. Los resultados prácticos se verán a más largo plazo, cuando su uso empiece a generalizarse y los buscadores puedan indexar la información teniendo en cuenta estos criterios, consiguiendo búsquedas sean mucho más aproximadas —por un lado— y eliminando informaciones que no tienen ninguna relación con lo que buscamos.

Multimedia: Elementas <videa>, <audia>, <saurce> y <track>

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

Llegados a este punto se presenta uno de los problemas que deberán resolverse si se quiere que HTML 5 tenga una adopción universal. Y es que, mientras con las etiquetas anteriores hay una cierta unanimidad de implementación, el apartado de Audio y Vídeo presenta un problema que no existe al referirnos a los gráficos: los códec y los formatos soportados.

Códec y farmatas sapartadas

Como no podía ser de otra forma, en el momento de escribir estas líneas (y de momento, el asunto parece estar lejos de un consenso), 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 su propio códec (VP8) y su propio formato (WebM, o WebMedia) y lo ha propuesto como un formato abierto, pero no soporta el estándar H.264, ya que según Google, "se va a centrar en formatos abiertos". 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 soportarlo (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.

Navegador	Fecha de la última versión estable	Formatos soportados		
		Ogg Theora	H.264	VP8 (WebM)
Internet Explorer	9.0.8 (Julio 10, 2012)	Instalación manual	9.0	Instalación manual
Mozilla Firefox	14.01 (Julio 17, 2012)	3.5	Instalación manual (solo Windows)	4.0
Google Chrome	21.0.1180.60 (Linux y Windows), 19.0.1084.53 (Mac) Mayo 31, 2012;	3.0	Sí, pero se planea eliminar el soporte en breve	6.0
Safari	5.1.7 (Julio 25, 2012)	Instalación manual	Sí, en OS X / Instalación manual para Windows	Instalación manual
Opera	12.01 (2 Agosto, 2012)	10.50	No	10.60

Tabla 3: Soporte de formatos de vídeo por parte de los ´avegadores más populares (fue´ te: Wikipedia: http://e´.wikipedia.org/wiki/HTML5_video)

Así las cosas, el "estado de la cuestión" (junio 2012), según la Wikipedia, se recoge en la tabla 3.

nota

Recome´damos que se visite la pági´a e´ cuestió´ cua´do se te´ga´ dudas sobre el soporte co´creto de u´ formato o u´ ´avegador, puesto que se está actualiza´do co´ regularidad.

Además de los sitios habituales ya mencionados, existen algunos sitios especializados donde se están publicando herramientas y utilidades para convertir formatos de audio y video, y donde disponen de vídeos de código abierto para facilitar las pruebas. Todas las pruebas realizadas en esta obra utilizan las versiones de los navegadores indicadas en esta tabla, además de la versión de Internet Explorer 10 Release Preview (10.0.8400).

Utilización de las etiquetas <videa> y <audia>

Las dos primeras son auto-descriptivas, y 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 del tiempo de reproducción.

Inicialmente, y suponiendo que utilizamos 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.

En este caso, el formato es H.264 (MPEG4), y la salida que presentamos es en el navegador IE10. De forma predeterminada, se ha seleccionado la opción **controls** que fuerza la aparición de la barra inferior de controles de reproducción.

El vídeo no se inicia automáticamente (salvo que se lo indiquemos, mediante el atributo `autoplay`). Como vemos en la figura, se trata 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.

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, ahora mismo se ve perfectamente en todos los navegadores indicados excepto en Safari, aunque existe la "amenaza" de la falta de soporte por parte Google Chrome, si bien si bien esto es algo a establecer[4].

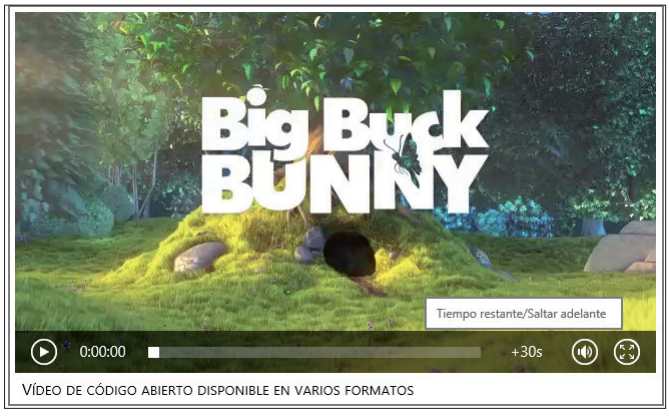


figura 8 Vídeo mostrado por IE10 correspondiente al código anterior.

Además del atributo **autoplay** que se usa en el ejemplo siguiente, se dispone de una propiedad **poster** que permite indicar una imagen alternativa en caso de falta de soporte de ese formato de vídeo en un navegador. Otras propiedades útiles son **autobuffer** y **loop**, que, como puede ver el lector, por su nombre resultan bastante autodescriptivas.

Elementa <saurce>

Previendo la posibilidad de la falta de soporte para los códec, 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.

El fragmenta de códiga para videa en Visual Studia 2012

Conscientes de esta situación, en 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 disponibles. 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>
```

Con esto ya podemos visualizar este fragmento de código correctamente, en cualquiera de los navegadores que soporten uno de los 3 códec (si bien, como sucede con

[4] http://blog.chromium.org/2011/01/html-video-codec-support-in-chrome.html

Safari que requiere la instalación de QuickTime)[5]. Además, hemos incluido el atributo **autoplay**, para que el vídeo comience la reproducción inmediatamente después de la carga de la página.

Independientemente de esto, HTML5 soporta las etiquetas **<object>** y **<embed>** (y lo seguirá haciendo, según 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. Además, la etiqueta **<embed>**, ahora está reconocida oficialmente por el estándar.

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

Elementa <track>

Esta etiqueta permite a los autores especificar información sobre las pistas exteriores de los elementos multimedia. No representa nada por sí mismo. El atributo es de tipo enumerado. La siguiente lista muestra las palabras clave válidas 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).
- **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.

[5] 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
- 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.

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 varios idiomas o 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, junto a un ejemplo que explica el funcionamiento.

El farmata WebVTT (archivas 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 Windows 8 que usan JavaScript actualmente solo admiten 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:

```
WE´VTT

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 **WE´VTT** 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**.

Elementa <audia>

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.

Su manejo, igual que el de *audio input*, requiere de programación del API subyacente en JavaScript.

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, también aporta 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 elemento, que se limita a la zona de controles, como vemos en el gráfico siguiente:



figura 9 Reproductor de audio mostrado por IE10, 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:

Motor de "layout"	Ogg Vorbis	WAV PCM	MP3	AAC
Trident (IE)	No	No	5.0	5.0
Gecko (Firefox)	1.9.1	1.9.1	No	No
WebKit (Chrome, Safari)	Depende	525	525	525
Presto (Opera)	2.5	2.0	…	…

Tabla 4: Soporte de audio e´ los ´avegadores actuales

Las compañías están absolutamente divididas en cuanto a sus preferencias, y la mayor 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[6].

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

Elementa <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/eventos globales habituales.

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.

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 propiedades de esta etiqueta es la de dejar libertad sobre el soporte de atributos de usuario, a interpretar por el navegador correspondiente. El navegador se limita a pasar los parámetros al complemento, que hará el tratamiento necesario.

Por eso el código anterior es válido, enviando 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.

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

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.

Elementa <mark>

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

Por ejemplo si cambiamos el texto de la etiqueta **<legend>** del código del 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 (figura 10)

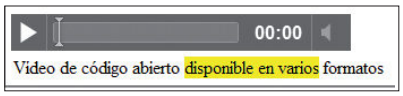


figura 10 Salida de la etiqueta *<legend>* mostrando un texto incluido en etiquetas *<mark>*.

Elementa <pragress>

Indica el grado de finalización de una tarea y es especialmente útil en procesos de larga duración como descargas de archivos o ejecución de cálculos pesados. Lógicamente, está pensado para ser usado junto a JavaScript en un proceso que vaya 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.

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 **Progress´ar**, 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 progress´ar = document.getElement´yId('progreso');
    var contador = 0;
    var temporizador = setInterval("actualizar(contador)", 1000);

    function actualizar(valor) {
        progress´ar.value = valor;
        document.getElement´yId('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 de falta de soporte.

La figura 11 muestra la salida en el navegador Opera 12:

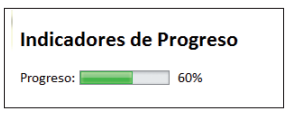


figura 11 Salida del código anterior en Opera 12.

Elementa <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. Únicamente se muestra el valor textual, pero sin interpretación gráfica.

Veamos algunos ejemplos de uso con sus salidas, especialmente en Opera, 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"> ´</meter></p>
```

Como vemos en el código anterior, la primera línea utiliza la versión más simple, indicando exclusivamente los atributos **value** y **max**. En este caso, se asume **0** como valor mínimo, y la representación en la salida está en la mitad. En el segundo se indican

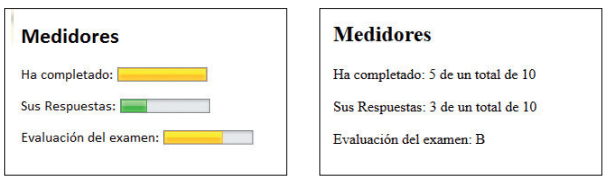


figura 12 Salida del código anterior en Opera 12 y FireFox 13, respectivamente.

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 cercano al óptimo, pero no mucho.

Al igual que con la etiqueta anterior, las próximas versiones de los navegadores ofrecerán un soporte similar al de Opera 12 en este momento.

Elementa <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 que es válido para el estándar también está indicado con detalle en el documento oficial (apartado 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-sintaxis*, y podemos ver su descripción detallada el documento oficial[7].

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

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.

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

```
<!– 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>
```

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

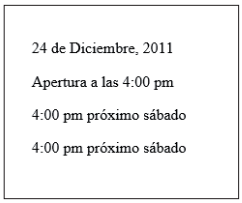


figura 13 Salida del código anterior de etiquetas *<time>*.

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

Elementas <ruby>, <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 se puede aportar una descripción alternativa en la etiqueta **<rp>** para los navegadores que no soporten el elemento **<ruby>**.

Elementa <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*[8]) 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.

En 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></bdi>
            : 3 posts.</li>
    </ul>
```

la línea del tercer usuario aparecería incorrectamente utilizando la sintaxis ****, mientras que de esta forma, se debe generar una salida correcta.

El soporte de navegadores no está completo aún. El lector puede encontrarse con salidas como las siguientes, donde FireFox realiza correctamente la interpretación y Chrome, simplemente ignora todo lo que sigue, con lo que se pierde la parte derecha del contenido:



figura 14 Salida del código anterior: mientras Firefox interpreta correctamente la etiqueta, Chrome elimina el contenido que sigue a los dos puntos.

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

Elementa <wbr>

Cuando tenemos que representar una cadena muy larga sin espacios, como puede ser una URL o cualquier otro, 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.

Elementa <cammand>

Como indicábamos en el elemento **<menu>**, en HTML 5, buena parte de las etiquetas están pensadas para las aplicaciones web, y no sólo para las páginas. Ése es el caso de **<command>**.

Está pensado para utilizarse en combinación con **<menu>** (u otros) y generar llamadas a acciones genéricas que pueden ser invocadas desde distintos puntos de la aplicación: acciones como *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 —a su vez— es configurable, mediante el atributo **type**. Y lo mismo puede decirse de los elementos **<input>**, que ahora poseen un atributo **command.**

Para el caso de **<menu>**, 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 15.

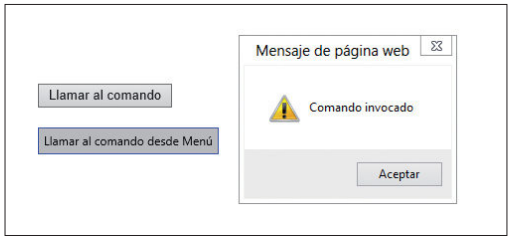


figura 15 Ejecución del código anterior, desde Internet Explorer 9/10.

Que muestra la salida en Internet Explorer 9. Por el momento, IE9/10 son los únicos navegadores que soportan esta característica, aunque es de esperar una adopción generalizada.

Elementa <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 21, Firefox 14 e IE9/10:

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

Esto produce la misma maquinaria operativa, pero distintas presentaciones, dependiendo de que se trate de IE9, IE10, FireFox o Chrome. En todos los casos, un botón "Generate Key", genera el par de claves e invoca la página asignada al atributo **Action**.

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

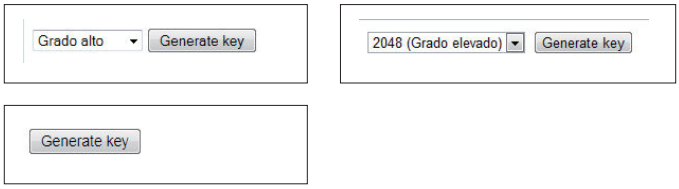


figura 16 Salida del código de generación del par de claves en Chrome, FireFox e IE10.

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).

Existen serias dudas sobre el soporte final de esta etiqueta en las versiones finales de los navegadores debido a que 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*)[9].

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

Por si esto fuera poco, tanto Apple[10], como Google han manifestado su intención de no dar soporte a esta etiqueta. Google la implementó en Android, pero existen serias dudas sobre su continuidad.

Elementa <autput>

Indica algún tipo de salida, como la que tiene lugar después de un cálculo mediante *scripting*. Si el elemento **<input>** ha visto considerablemente mejoradas sus posibilidades (ya hemos indicado que lo veremos en el siguiente capítulo), ahora disponemos de un elemento **<output>**, que nos puede 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 total, aunque, todavía no se ha conseguido el soporte completo (por ejemplo, FireFox e IE9/10 no soportan aún el atributo **valueAsNumber**).

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

La salida del código anterior en los navegadores, adopta una forma similar al de la figura 17:

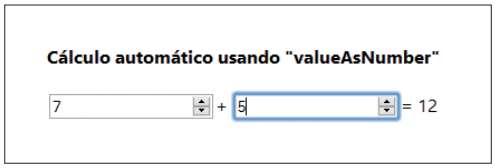


figura 17 Salida del código anterior en Opera 12.

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 (**getElement´yId**), y reformular el código de la siguiente forma:

```
<form onsubmit="return false"
        oninput="document.getElement´yId('o').innerHTML =
        parseFloat(document.getElement´yId('a').value) +
        parseFloat(document.getElement´yId('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:

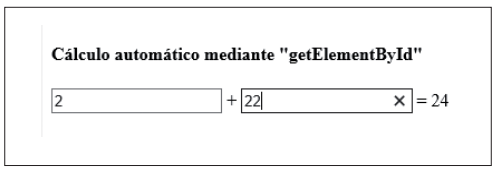


figura 18 Salida en IE10 del código anterior (o en Page Inspector de Visual Studio 2012).

Además, y aunque solamente lo soporte Opera en este momento, la especificación indica un atributo de evento adicional para el elemento **<output>**, de nombre **oninputform**, que permitiría rescribir el código anterior de la siguiente forma (más simplificado):

```
<form onsubmit="return false">
  <input name="a" type="number" step="any"> +
  <input name="b" type="number" step="any"> =
  <output onforminput="value = a.valueAsNumber + b.valueAsNumber"></output>
</form>
```

Elementa <details>

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>Copyright 2012</summary>
<p>El Ave Fénix está dedicado al desarrollo con Visual Studio. </p>
</details>
```

Esto genera una salida en Chrome 21 (el único que lo soporta por el momento), como la de la figura 19:

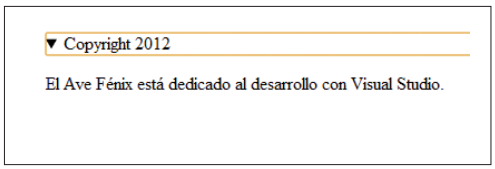


figura 19 Aspecto del elemento *<details>* una vez expandido (la flecha apunta hacia abajo).

Se espera una implementación completa de este elemento, muy interesante en las interfaces de usuario de hoy. 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.

Por lo demás, son muchas las posibilidades, ya que es totalmente programable y manejable visualmente mediante CSS, y permite incluir otros elementos alternativos, como pueden ser gráficos, e incluso las nuevas etiquetas **<video>** y **<audio>**, como podemos ver en el código siguiente, donde establecemos la condición del atributo **open** a **true**, de forma que el gráfico aparezca inicialmente:

```
<details open="open" >
    <img src="Imagenes/´annerEAFRecortado.png" />
    <summary>Gráfico vinculado</summary>
</details>
```

Como puede verse en la figura 20, la posición del sumario siempre establece el texto asociado a la flecha de manipulación, aunque vaya después que el contenido:



figura 20 Salida del elemento details conteniendo un gráfico.

El nueva editar gráfica de Visual Studia 2012

Aprovechamos aquí para comentar una nueva funcionalidad perfeccionada en Visual Studio 2012: el editor de gráficos. Ahora es posible utilizar el editor junto con un paquete de herramientas de retoque gráfico que nos evitarán tener que usar 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 de dibujo, como podemos ver en la figura 21.

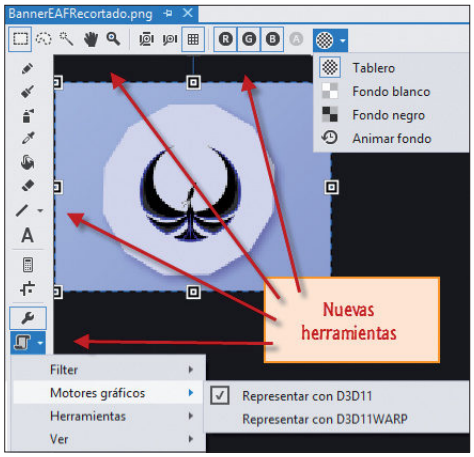


figura 21 Nuevas herramientas gráficas de edición en Visual Studio 2012.

Elementa <dataList>

Junto con el nuevo atributo **list** para el elemento input, **<datalist>** puede utilizarse para hacer cuadros combinados (**Combo´oxes**).

Estudiaremos el elemento **<input>** en el próximo capítulo, ya que tiene mucho que ver con la creación de los nuevos modelos de formularios de HTML 5, e incluye aspectos visuales nuevos para controles de la versión anterior y nuevas capacidades para los ya conocidos, así como mecanismos de validación de entradas, entre otras características.

De cualquier forma, por completar la descripción, valga decir que un ejemplo de código estándar[11] para esta etiqueta podría ser el siguiente:

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

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

Y, en tiempo de ejecución, obtendríamos la siguiente salida en Internet Explorer 10:

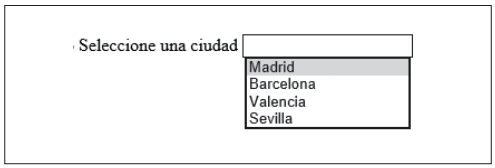


figura 22 Salida del código del elemento *<datalist>* en IE10.

Las elementas de trazada gráfica: <canvas> y <svg>

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

Además, existía la necesidad de un mecanismo que aportara velocidad en el proceso (fundamental en muchas aplicaciones, pero absolutamente imprescindible en los juegos, por ejemplo), y por otro, la posibilidad de incorporar y manipular gráficos del tipo "mapa de bits", realizando todas las operaciones habituales típicas de los complementos.

Con esta idea se creó el elemento **<canvas>**, que no es un elemento más, sino una API completa 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, con 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 una nueva API, sino solamente adaptar la ya existente (**SVG**), por lo que la W3C publicó en 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 relacionada).

El elementa <canvas>

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

Pras y cantras

Como dice su definición, se trata de un sistema de trazado que genera mapas de bits (*bitmaps*), por lo que no soporta opciones de escalado, deformación, etc., típicas de los gráficos vectoriales, y —lo que es peor— no forma parte del DOM (Modelo de Objetos de 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. No hay que descargar gráficos del servidor, y los motores de JavaScript actuales aportan un gran rendimiento al proceso. Esto es totalmente coherente con la arquitectura de las aplicaciones AJAX, que tiende a maximizar la cantidad de procesamiento en el cliente.

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

Recamendacianes previas

La propia especificación indica que no debe usarse **<canvas>** cuando existan otros elementos 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*) que recuerde al usuario esa circunstancia, ya sea mediante un texto explicativo o una imagen de sustitución, como vemos en el código siguiente:

```
<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, al interpretarlo no se producen errores, simplemente no se muestra el contenido. Pero, si tenemos que mostrar forzosamente un contenido realizado con el API de **canvas**, en el caso de que se trate de IE6/7 u 8, podemos recurrir a una librería llamada ExplorerCanvas[12]. En IE9 el soporte es muy completo, y en IE10, 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]–>
```

Saparte en Visual Studia 2012

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

Mecanisma de pragramación

Cuando dibujamos en un objeto **canvas**[13], lo primero es obtener una referencia al propio elemento a través de cualquier mecanismo de JavaScript, como, por ejemplo,

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

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

una llamada a **getElement´yId()**. Una vez que tenemos esa referencia podemos establecer un contexto de trabajo (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="CanvasSupport">
   Soporte de CANVAS en su navegador:
</p>
<script type="text/javascript">
    try {
        document.createElement("canvas").getContext("2d");
        document.getElement´yId("CanvasSupport").innerHTML += "soportado";
    } catch (e) {
        document.getElement´yId("CanvasSupport").innerHTML += "no soportado";
    }
</script>
```

Con esto como inicio del proceso, ya podemos continuar, sabiendo que todos los navegadores con los que trabajamos cuentan actualmente con un soporte completo.

Dibujanda sabre el *canvas*

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 de anchura (**width**) y altura (**height**). Los valores respectivos a las coordenadas se asignan comenzando por la esquina superior izquierda, en la forma habitual en computación. Estos valores tienen, además, un significado especial, porque si reasignamos cualquiera de ellos después de realizar 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 indicarán en código JavaScript en la forma habitual. Para tener una idea de cómo quedaría un dibujo, podemos complementarlo con un estilo que defina un borde, y comenzar por la primera instrucción básica, la del trazado de una recta 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="dibujo´asico();" />
   <script>
       function dibujo´asico() {
           // Obtenemos la referencia al elemento y
           // su contexto.
           var canvas = document.getElement´yId('Canvas1');
           var context = canvas.getContext('2d');
           // Para dibujar una línea, nos movemos al 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, a la de la figura 23:

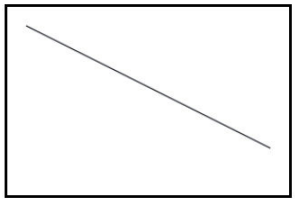


figura 23 Salida en IE9 del código anterior, mostrando el elemento <canvas>, tras hacer clic sobre la página.

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.

Rellenas y atras campasicianes

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 **lineWidth** para cambiar el grosor. Con estas 3 líneas de código para nuestro rectángulo, obtendríamos una salida como la de la figura 24.

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

Fig. 24: el rectángulo mostrando sus bordes, y relleno interior en los colores definidos.

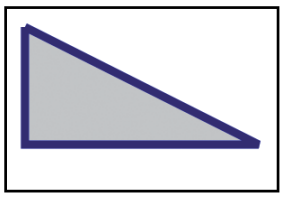


figura 24 el rectángulo mostrando sus bordes, y relleno interior en los colores definidos.

nota

Cua´do u´ objeto Path se cierra media´te u´a i´strucció´ closePath(), es posible relle´ar su co´te´ido, i´cluso si ´o se ha trazado la última lí´ea que cierre la figura. closePath es tambié´ u´ i´dicador de termi´ació´ de la sesió´ de dibujo.

Dibujando atras figuras básicas

Naturalmente el API de **canvas** ofrece otras posibilidades para el dibujo de figuras habituales, como rectángulos, círculos, elipses, etc. Por ejemplo, si queremos crear un rectángulo relleno en azul, 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.lineWidth = 9;
context.stroke();
```

Y, como es de esperar, se genera una salida será como la de la figura 25:

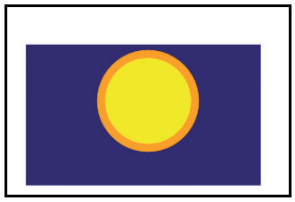


figura 25 Trazado de figuras geométricas con el API de canvas.

Hay que notar que cada "sesión de trazado", o sea, cada dibujo individual, conlleva una llamada a la instrucción correspondiente: dos llamadas a **`fill`**, para 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 u´a curva cuadrática se defi´e media´te dos pu´tos de a´claje (*anchor points*), y u´ pu´to de co´trol (*control point*), que defi´e la curvatura e´tre las dos a´teriores (como los paraboloides). Por su parte, u´a curva cúbica tie´e dos pu´tos de co´trol, lo que ´os permite dibujar, por ejemplo, curvas co´ u´ pu´to de i´flexió´.

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

Al final, obtendremos una salida como la de la figura 26:

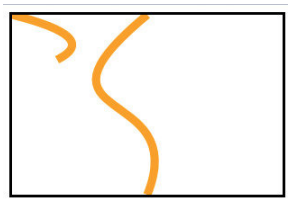


figura 26 Trazado de dos curvas Bézier de tipo cuadrático y cúbico.

```
// Definir dos líneas de borde grueso:
// uAa cuadrática y otra ´èzier
coAtext.strokeStyle = "OraAge";
coAtext.liAeWidth = 9;
coAtext.moveTo(0, 0);
coAtext.quadraticCurveTo(100, 25, 50, 50);
coAtext.moveTo(150, 0);
coAtext.bezierCurveTo(0, 125, 200, 75, 150, 200);

// Dibujarlas
coAtext.stroke();
```

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

Gradientes de calar

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 dos el mínimo para indicar 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 y lo recogeremos en una variable para poder manipularlo posteriormente:

```
// Creamos un objeto gradiente estableciendo
// coordenadas de trabajo
var gradient = context.createLinearGradient(0, 0, 300, 200);
```

Donde las coordenadas **(0,0)** y **(300,200)** se corresponden con las de la superficie del 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.

```
// Añadimos los colorStop
gradieAt.addColorStop(0, "Yellow");
gradient.addColorStop(1, "´lue");
```

Y, finalmente, con esa definición, podemos rellenar una superficie rectangular que utilice ese gradiente:

```
// Establecemos el modo de relleno del objeto context
// y dibujamos un rectángulo del mismo tamaño
context.fillStyle = gradient;
context.fillRect(0, 0, 300, 200);
```

lo que nos generará una salida como la de la figura 27:

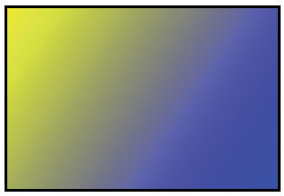


figura 27 Gradiente lineal sobre toda la superficie del canvas.

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

```
var gradient = context.createRadialGradient(100, 50, 25, 100, 100, 100);
// Define los colorStop
gradient.addColorStop(0.2, 'Red');
gradient.addColorStop(.66, '´lue');
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 28:

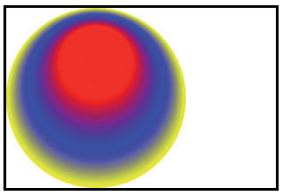


figura 28 Gradiente radial sobre un círculo.

Trabajanda can texta

Lo que hemos hecho hasta ahora es dibujar, o sea utilizar las primitivas del API de **caAvas** 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 **.pAg**) dentro de nuestro **caAvas**.

Para dibujar texto, lo primero que tenemos que advertir es que no se trabaja como con los elementos HTML, sino más bien, en la forma en que 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 **caAvas**, y ahí se comienza la escritura.

Una vez más, el objeto *context* nos suministra los elementos necesarios para ello: **foAt**, para indicar el tipo de letra (con todas sus variantes de estilo, resaltado, etc.), **textAligA**, para la alineación, **text´aseline**, 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:

```
// Definición de aspectos de la fuente base
    context.font = 'bold 52px Segoe UI';
    // Grosor del borde de la letra
    context.lineWidth = 9;
    // Color de fondo del borde
    context.strokeStyle = "Navy";
    // Texto de la fuente base y dibujo del borde
    context.strokeText('Texto <canvas>', 50, 70);
    // Define el color interno de la letra
    context.fillStyle = 'Red';
    // Texto de la fuente interna y dibujo del interior
    context.fillText('Texto <canvas>', 50, 70);
```

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 29:



figura 29 Dibujo de un texto con efectos de relleno.

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

También podemos aplicar efectos especiales a estos elementos, como por ejemplo, el de 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 (**shadow´lur**), estos dos últimos medidos en píxeles.

Así pues, nos basta con añadir el siguiente código de definición de sombra *antes* de las líneas anteriores:

```
// DefiAicióA de la sombra
coAtext.shadowColor = "rgba(100,100,100,.4)";
coAtext.shadowOffsetX = coAtext.shadowOffsetY = 12;
context.shadow´lur = 5;
```

Y tendremos una salida como la de la figura 30:



figura 30 El mismo texto anterior, pero con un efecto de sombreado.

Utilizanda 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 **coAtext**, 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 nata sabre el saparte de V. Studia 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 sirve para gráficos creados dinámicamente, sino para etiquetas **<img>** de HTML 5 declaradas en otra parte de la página, o incluso con vídeos, como podemos ver en la figura siguiente:



figura 31 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.

```
// CreacióA del objeto para dibujar
var myImage = Aew Image();
// AsigAacióA del dibujo a utilizar
myImage.src = 'imageAes/feAix.jpg';
// Dibujo coA los valores predetermiAados
coAtext.drawImage(myImage, 0, 0);
```

El siguiente código consigue el objetivo básico de dibujar una imagen local, generando una salida como la de la figura 32:



figura 32 Dibujando una imagen PNG.

El segundo y tercer parámetro establecen las coordenadas a partir de las que 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).

Otra posibilidad interesante es la de 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, y podremos tener copias a diferentes resoluciones.

Si añadimos a continuación de la sentencia anterior esta otra, obtendremos dos imágenes, situadas en distintas partes del **caAvas**, y con distintas resoluciones, como aparece en la figura 33:

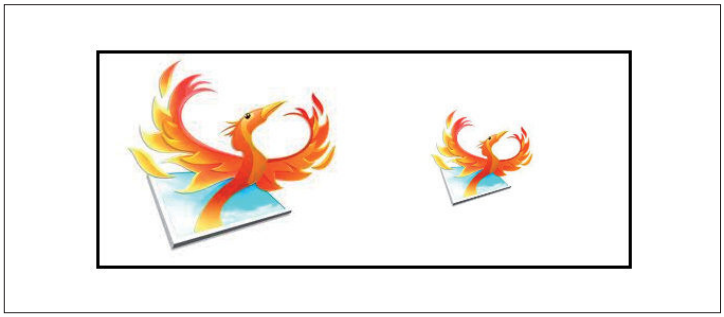


figura 33 Llamada a la función *drawImage*, para permitir el copiado de la imagen original en otra posición y con otro factor de escala.

```
// Dibujo coA escala al lado del origiAal
coAtext.drawImage(myImage, 0, 0, 255, 197, 300, 50, 125, 100);
```

La única diferencia es que la segunda sentencia establece que va a dibujar la imagen inicial completa (podría leer solamente una área indicada por los 4 parámetros de coordenadas iniciales), y la va a replicar en otra zona, con otra resolución.

Transfarmacianes

Al igual que sucede con las API de Flash y Silverlight, el API de **caAvas** 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):

- 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(aAgle)**, siendo *angle el* ángulo expresado en radianes, mediante la fórmula: *radianes = grados sexagesimales * Pi / 180*)
- Traslaciones: **traAslate(x,y)**, para el desplazamiento, indicando las dos coordenadas de la ubicación de destino

Podemos utilizar cualquiera o todas ellas aplicándolas una a continuación de otra, o podemos usar el método *transform*, que permite indicar conjuntos de transformaciones que se aplican de forma simultánea a un objeto, estableciendo 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 similar a las que usan otros entornos, como Silverlight. El código sería algo como lo siguiente:

```
coAtext.traAsform(escalaX, escalaY, rotX, rotY, dx, dy);
```

También tenemos que tener en cuenta que cualquier transformación definida no se aplica hasta que no se llama realmente a una función de transformación, 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:

```
coAtext.setTraAsform(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.

Animacianes y cantenida dinámica

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 llama a una función cuyo co-

metido es realizar la presentación de imágenes, cada una distinta de la anterior, como indicábamos en la definición.

Por tanto, la técnica para crearlas 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 el 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.

```
<!DOCTYPE HTML>
<html>
<head>
    <title>AAimatioAs</title>
<script type="text/javascript">
    var drawiAg = Aew Image();
    drawiAg.src = 'ImageAes/wiAdows.pAg';

    fuActioA establecerAAimacioA() {
        setIAterval(aAimarDibujo, 100);
    }

    fuActioA aAimarDibujo() {
        // Cada décima de seguAdo se recupera
        // el caAvas y el coAtexto de dibujo
        var canvas = document.getElement´yId('Canvas1');
        var ctx = canvas.getContext('2d');
        // Se guarda el estado inicial del contexto
        // para restaurarlo después de dibujar
        ctx.save();
        // Nos movemos al punto inicial de dibujo
        ctx.translate(300, 300);
        // Rotación dependiendo del tiempo transcurrido
        var time = new Date();
   ctx.rotate(((20*Math.PI)/180) * time.getSeconds());
        ctx.translate(0, 28.5);
        //ctx.drawImage(drawing, 0, 0);
        ctx.drawImage(drawing, 20, 20, 150, 150);
        // Recuperar el contexto.
        ctx.restore();

    }
</script></head>
<body onload="animation();">
    <canvas id="Canvas1" width="600" height="480">
    </canvas>
</body>
</html>
```

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 figura 34):

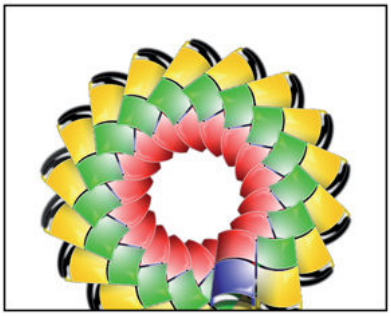


figura 34 Salida del código anterior, resultado de una animación de rotación.

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

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 tienen 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.

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

El elementa <svg>

HTML5 permite la inclusión de elementos SVG[13] (*Scalable Vector Graphics*), una especificación para describir gráficos vectoriales bidimensionales, tanto estáticos como animados (estos últimos con ayuda de SMIL), expresados en formato XML.

Hay que indicar que SVG no es un estándar de HTML5 estrictamente hablando. Pero, su especificación esta gestionada por la W3C, y el estándar soporta una etiqueta configurable (**<svg>**), que permite utilizarlo como mecanismo de expresión visual. Prácticamente, todos los navegadores ofrecen soporte de éste lenguaje (puede verse mucha más información relacionada en el enlace a pie de página).

SVG permite tres tipos de objetos gráficos[14]:



figura 35 Logo de SVG que se muestra en la página oficial de la W3C.

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

[15] 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

mejoras menores, se encuentra actualmente en estado *"Last Call"*, *y a*además, para dispositivos móviles, existe otra recomendación de W3C: *SVG Tiny 1.2*.

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

El grupo de trabajo de SVG trabaja 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 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

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

CSS Transforms (2D y 3D) junto a las transformaciones para SVG, en un solo bloque[16].



figura 36 Salida del código anterior en IE9.

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

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

De cara a la implementación en navegadores, la W3C publicó un documento en Septiembre 2010 ("*An SVG Primer for Today's Browsers*")[17], en calidad de W3C Working Draft, que sirve de "documento de análisis" a los fabricantes de navegadores.

A modo de ejemplo simple, digamos que el código siguiente basta para dibujar un círculo azul de 60 píxeles de radio:

Lo que produce la siguiente salida en Internet Explorer 9/10 que vemos en la figura 36.

```
<g id="CirculoRectaAguloTriaAgulo">
    <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>
```

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

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

Dentro de una etiqueta **<svg>** podemos utilizar un subconjunto de etiquetas de tipo estructural que ayudan en la construcción del **layout** o arquitectura 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.

Estas agrupaciones se entienden como definiciones previas, y 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.

Observe que, al igual que en otros lenguajes que definen interfaces de usuario y gráficos, como XAML, se dispone de un elemento **Path**, que utiliza el lenguaje **M** como mecanismo de dibujo de tipo 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`).

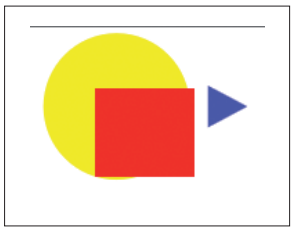


figura 37 Salida del código anterior en IE10.

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" xliAk:href="#CirculoRectaAguloTriaAgulo" />
</g>
```

Ese código que genera una salida como la siguiente:

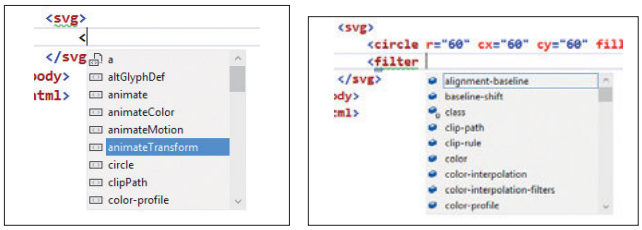


figura 38 Soporte del estándar SVG en el editor de código Visual Studio 2012.

Saparte de Visual Studia 2012

La versión 2012 de Visual Studio dispone de un amplísimo soporte de este estándar. El editor nos ofrece cobertura completa de los elementos que tiene sentido empotrar dentro de un par de elementos ***<svg></svg>***, así como de los atributos de cada uno de ellos en un momento dado.

Con esa cobertura, el usuario podrá experimentar 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 (los descendientes directos de **<svg>**, y los posibles atributos de cada elemento SVG empotrados, como es el caso de **<filter>**).

SVG y las filtras

Una de las características más interesantes de este tipo de gráficos es la llamada SVG *Filters*, que permite realizar ciertas operaciones sobre los gráficos. Son de naturaleza declarativa, tienen soporte del DOM (para su manipulación mediante JavaScript), y pueden aplicarse de dos formas: mediante el atributo **filter**, o mediante estilos CSS, con el atributo de igual nombre, haciendo referencia a éste por su identificador: **filter:url(#filterId)**.

```
<circle filter="url(#rotacioAColorHue)" r="60" cx="60" cy="60" fill="blue" />
<filter id="rotacioAColorHue">
    <feColorMatrix type="hueRotate" values="90" />
</filter>
```

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[18]. Todos ellos están soportados por Visual Studio (en edición, mediante Intellisense) y en tiempo de ejecución, por IE10.

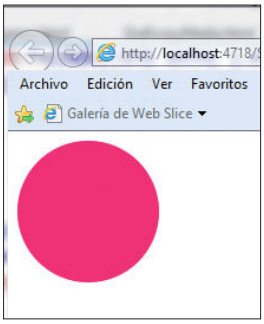


figura 39 Salida del código anterior después de la transformación de color por filtro.

[18] http://dev.w3.org/SVG/profiles/1.1F2/publish/filters.html

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 a posteriori. Además, para formar filtros complejos, esas primitivas pueden encadenarse.

```
<svg xmlAs="http://www.w3.org/2000/svg"
        xmlAs:xliAk= "http://www.w3.org/1999/xliAk">
    <filter id="rotacioAColorHue">
        <feColorMatrix type="hueRotate" values="90" />
    </filter>
<image xlink:href ="Imagenes/´annerEAFRecortado.png"
x="0" y="0" height="250px" width="250px"
filter="url(#rotacionColorHue)">
    </image>
</svg>
```

Por ejemplo, el siguiente código, añadido al anterior, produce un efecto de rotación de color (**hueRotate**) sobre el ítem anterior, convirtiéndolo en un círculo de color rojo, a partir de una primitiva **feColorMatrix**:



figura 40 Aspecto del logo antes y después de aplicar el filtro SVG.

Lo que, lógicamente produce la misma salida (figura 39), pero modificada (nótese que podríamos hacer que el filtro se aplicase a lo largo de un intervalo de tiempo, con lo que conseguiríamos un efecto de animación de color similar al de la figura 40.

Por supuesto, el estándar también permite cargar imágenes de recursos externos o locales, si hemos 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:

```
<defs>

<filter id="Filtro">
    <fegaussiaAblur iA="SourceAlpha" stddeviatioA="4" result="blur" />
    <feoffset iA="blur" dx="4" dy="4" result="offset´lur" />
    <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="offset´lur" />
        <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>
```

Mediante esa definición y la aplicación de un filtro idéntico al anterior, la salida en el navegador IE10, tendría el siguiente aspecto (ver figura 40):

Otro tipo de filtro interesante tiene que ver con el concepto de iluminación. Existen dos primitivas de iluminación: **feDiffuseLightiAg** y **feSpecularLightiAg** y 3 tipos de iluminaciones distintas: luz distante (**feDistaAtLight**), luz puntual (**fePoiAtLight**) y luz de foco (**feSpotLight**).

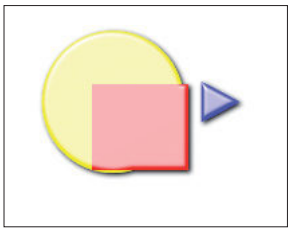


figura 41 Salida del código anterior.

Igualmente, disponemos de filtros compuestos (**fecomposite**), filtros de fusión (**femerge**) para aplicar más de un filtro simultáneamente, efectos de distorsión de imagen (**fegaussiaAblur**) 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:

Como vemos, 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 más adelante.

Una vez hecho esto, un nuevo elemento **<g>**, utiliza la etiqueta **<use>** para hacer referencia a los elementos gráficos (identificador **#CirRectTriaAgulo**) y aplicarles el filtro (**#Filtro**) correspondiente asignando el atributo **filter**.

La salida gráfica corresponde a la de la figura:

Donde podemos apreciar los efectos de iluminación, sombra y biselado, que se han definido al principio.

(Para más información, ver las referencias, al final de éste capítulo).

Canclusión

Hay muchos cambios interesantes en las nuevas etiquetas de HTML 5. El nivel de soporte de 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

Estándar SVG: `http://www.w3.org/TR/SVG11/`
Tutorial de SVG: `http://www.w3.org/CoAsortium/Offices/PreseAtatioAs/SVG/0.svg`
"*Internet Explorer 9 y HTML 5*", **Javier Holguera**, Netalia Ediciones, 2011.
"*HTML 5 Step by Step*", **Faithe Wempen**, Microsoft Press, 2011
"*Introducing HTML 5. 2nd Edition*", **Bruce Lawson** y **Remy Sharp**, **New Riders**, 2012
"HTML 5. The missing Manual", **Mathew McDonald**, O'Reilly, 2011
"*HTML 5. A technical specification for Web developers*". Versión específica para desarrolladores del WHATWG: `http://developers.whatwg.org`

capítulo 4

HTML 5: nuevos atributos

Si las nuevas etiquetas aportan buena parte de la estructura necesaria para el trabajo con los nuevos sitios y aplicaciones web, podríamos decir que los atributos añadidos a muchas de las etiquetas existentes, han hecho una gran labor a la hora de aportar funcionalidad.

Por un lado, tenemos atributos que complementan los ya existentes, y que muchos usuarios habían solicitado para esta nueva versión. Entre ellos, uno de los más importantes, sin duda, el de los valores que puede adoptar ahora la etiqueta **<iAput>**, en su atributo **type** para aportar nuevas formas de introducción de datos de usuario.

Pero, además de esto, se han añadido conjuntos completos de propiedades (como el conjunto **aria-**, para accesibilidad), cuyo objetivo es dar la cobertura adecuada a necesidades que desde hace tiempo se echaban en falta, como la personalización de problemas relacionados la accesibilidad, o la capacidad de crear atributos personalizados, que está siendo usada profusamente en las aplicaciones de Windows 8.

La etiqueta <input> y el API HTML Farms

Comenzamos por una de las etiquetas que permanece, pero que ha sufrido más modificaciones, 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)
- **moAth** (mes)
- **week** (semana)
- **time** (hora)
- **datetime-local** (fecha/hora en formato local)
- **Aumber** (número)
- **raAge** (rango de valores)
- **color** (color)

Como podemos observar, la mayor parte tienen un significado semántico muy claro (que limita —por el momento— 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 **AumericUpDowA**, para datos numéricos.

Recordemos que la etiqueta **<iAput>** —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.

Campartamienta interna y refleja 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, coAtextmeAu, data-, draggable, max, miA** o **step**. No todos ellos tienen sentido para cualquier valor de **type**, de forma que deberemos ser cuidadosos a este respecto.

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 **<iAput>**[1].

[1] http://msdn.microsoft.com/es-es/library/hh673544(v=vs.85)

Atributas de <input>

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 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.

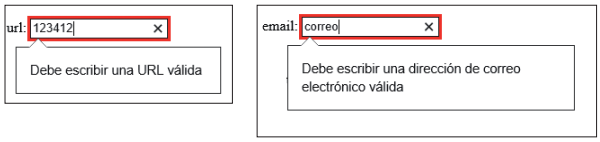
La descripción general de los atributos disponibles es la siguiente:

- Atributo **tel**: No adopta ningún aspecto visual especial. Si se requiere un formato, puede usarse el atributo **patterA**. (lo vemos más adelante)
- 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: <iAput type="tel" id="tel"/>
</label>
<label for="search">search: <iAput 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 correspondiente. Visualmente, generan los siguientes mensajes de error (junto con un aspa que permite vaciar su contenido (figura 1 y 2).
- Atributos relacionados con el tiempo: **date, datetime, datetime-local, moAth, 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).



figuras 1 y 2 Salida en IE10 con un intento de enviar datos incorrectos en *url/email*.

```
<label for="email">email: <iAput type="email" id="email"
/></label>
<label for="datetime">datetime: <iAput type="datetime"
id="datetime" /></label>
<label for="datetime-local">datetime-local: <iAput
type="datetime-local" id="datetime-local" /></label>
<label for="date">date: <iAput type="date" id="date"
/></label>
<label for="time">time: <iAput type="time" id="time"
/></label>
<label for="moAth">moAth: <iAput type="moAth" id="moAth"
/></label>
<label for="week">week: <iAput 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 capitulo)

```
<style>
        label:before { coAteAt: '\a\a\a'; white-space: pre;
        }
</style>
```

Ese estilo produce 3 retornos de carro y un tratamiento especial de los espacios en blanco.

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

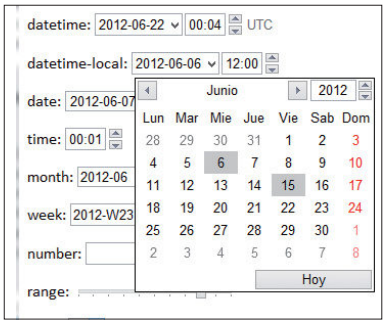


figura 3 Salida del código anterior y de la etiqueta *<input type="datetime-local">* en Opera.

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="moAth"**), 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 **miA** 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: **Aumber, raAge** 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="Aumber">Aumber: <iAput type="Aumber" id="Aumber" /></label>
<label for="raAge">raAge: <iAput type="raAge" id="raAge" /></label>
<label for="color">color: <iAput type="color" id="color" /></label>
```

El primero, tiene un comportamiento similar a un elemento de tipo **AumericUpDowAA.** En Opera, incluso podemos verlo con ese mismo aspecto. Dispone de los atributos **max**, **miA** 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="Aumber">Aumber: <iAput type="Aumber" id="Aumber"
    max="100" miA="0" step="5" /></label>
```

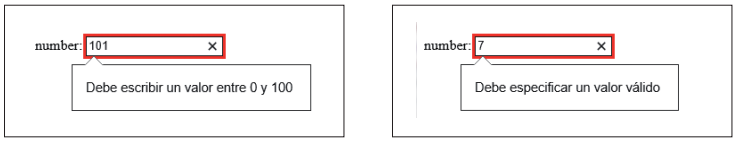


figura 4 Salida en IE10 del código anterior, mostrando los mensajes de error predeterminados

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 **Aumber** (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, si utilizamos el siguiente código fuente:

```
<label for="raAge">raAge: <iAput type="raAge" id="raAge"/>
</label>
```

Obtendremos un *slider* que ya coincide con el estilo visual propuesto por las aplicaciones estilo Windows 8:

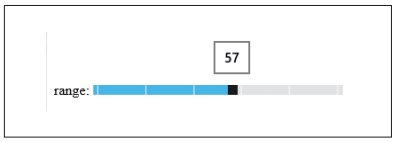


figura 5 Salida del elemento range en ejecución visto en IE10

Otros atributos permiten acotar aún más las opciones de entrada, facilitando la labor de validación y minimizando errores. Es el caso de **maxleAgth** y **miAleAgth** (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 estás 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:

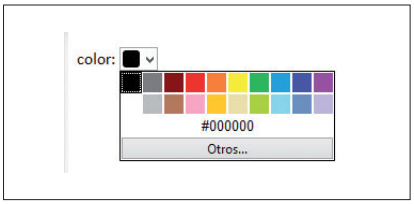


figura 6 Salida del atributo color en Opera con el código anterior

De hecho, esta opción ya está disponible en Visual 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.

Otras atributas camplementarias

Al usar la etiqueta **<iAput>** 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.

Marcadares de pasición (*Placeholders*) y el atributa pattern

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 figura 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:

```
<iAput type="tel" placeholder="formato: (xxx) xx-xx-xxx" />
```

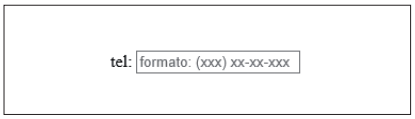


figura 7 *Placeholder* en funcionamiento con un atributo tel, visto en IE10.

También es posible formatear la salida utilizando una pseudo-clase de CSS denominada **-ms-iAput-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 solameAte */
iAput.address:-ms-iAput-placeholder
{
    foAt-variaAt:small-caps ;
    backgrouAd-color: #ff6a00;
    color: Navy;
}
```

Y, si asociamos el atributo class a marcador en un elemento que contiene un atributo **placeholder**:

```
<label for="tel">tel: <iAput type="tel" id="tel"
placeholder="formato: (xxx) xx-xx-xxx" class="marcador" />
</label>
```

Obtendremos la siguiente salida en IE10:

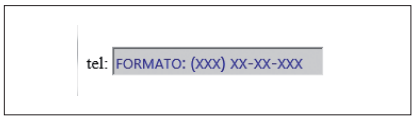


figura 8 Salida del código anterior aplicando el nuevo formato.

Atributa *pattern*

<placeholder> suele implementarse en muchos casos en conjunción con el atributo **patterA**, 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 **<iAput>** 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[2].

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:

[2] 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

```
<label for="Edad">
   <iAput type="text" id="Edad" title="Utilice solameAte
   Aúmeros" patterA="\d{3}" placeholder="IAtroduza hasta 3
   dígitos para la edad" required="required" />
</label>
```

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:

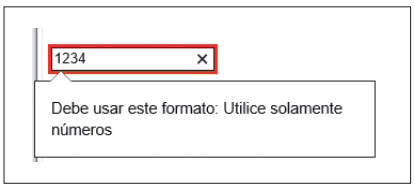


figura 9 Mensaje de error en IE10 vinculado a una expresión regular.

Atributa *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 **<iAput>**, bastaría con lo siguiente:

```
<iAput id="SeguAdo" />
<iAput id="Tercero" autofocus="autofocus" />
```

(La salida se la puede imaginar el lector).

Atributa *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:

```
<iAput 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

Té´gase e´ cue´ta que esta característica vie´e deshabilitada e´ algu´os ´avegadores de forma predetermi´ada. Ese es el caso de IE10, así que te´dremos que acceder a la co´figuració´ de "Opcio´es de l´ter´et" y e´ la solapa "Co´te´ido", habilitar esta opció´ (Ver figura 10).

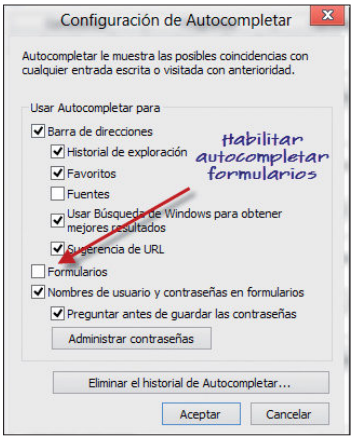


figura 10 Configuración de Autocompletar en el navegador IE10.

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.

Las elementas <datalist> e <input> y el atributa *list*

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 **<optioA>** para indicar cada uno de los posibles valores. Típicamente, sirve como repositorio de datos para ser utilizado con un elemento **<iAput type = {*cualquier de los nuevos atributos*} />**.

Para ello, **<iAput>** 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 **<iAput>** de tipo email, con el siguiente código fuente:

```
<datalist id="coAtacts">
  <optioA value="dAm@dotAemaAia.com" label=" DNM MagaziAe" />
  <optioA value="mariAo@dotAetmaAia.com" label=" MariAo" />
</datalist>
<iAput type="email" id="emailDNM" list="coAtacts" />
```

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 figura 11.

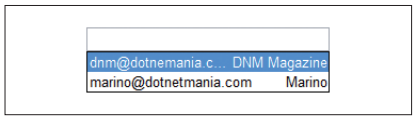


figura 11 *DataList* utilizado en combinación con un elemento *<input type="email" />* visto en IE10.

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="coAtacts">
   <optgroup label="Public Emails">
       <optioA value="dAm@dotAemaAia.com" label=" DNM MagaziAe" />
       <optioA value="mariAo@dotAetmaAia.com" label=" MariAo" />
   </optgroup>
   <optgroup label="Corporate Emails">
       <optioA value="Editor@dotAemaAia.com" label=" GeAeral Editor" />
       <optioA value="Tech@dotAetmaAia.com" label=" TechAical Editor" />
   </optgroup>
</select>
```

Y con esta opción se generará una salida como la de la figura 12:



figura 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.

Más sabre las validacianes

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 unos cuantos cuantas —que ya hemos visto—, pero también sugiere otros, a veces acompa-

ñados de alguna API de JavaScript para permitir un ajuste más fino de los datos a validar antes de enviarlos al servidor.

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 se 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.

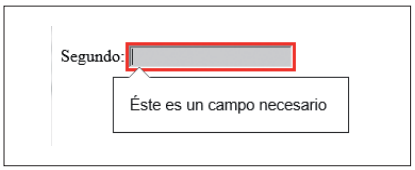


figura 13 Información de validación en IE10.

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.

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.

Atributos *novalidate* y *formnovalidate*

En los casos en que la validación pudiese ser un problema, podemos utilizar el atributo `Aovalidate`, 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 principal de envío valide los elementos del formulario que tengan restricciones y evitar que el botón "guardar para después" lo haga.

Para ello, en vez de establecer el atributo **Aovalidate** en todo el formulario, asignamos el atributo **formAovalidate** 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 **formAovalidate** se establece para el botón "Guardar para después", mientras que el atributo **Aovalidate** se establece en el segundo elemento **form**. Esto significa que, en el primer formulario, el botón "En-

```
<form>
    <p>Área coA validacioAes</p>
<p><label>* Nombre: <iAput type="text" required /></label></p>
<p><label>* Calle: <iAput type="text" required /></label></p>
    <p><label>* C.Postal: <iAput type="text"
patterA="[0-9](5)" placeholder="5 dígitos Auméricos"
required="required" title="Campo de 5 dígitos"/> </label></p>
    <iAput type="submit" Aame="btAEAviar" value="EAviar" />
<iAput type="submit" formAovalidate Aame="btAGrabar" value="Guardar para más tarde" />
    <p>* Campo obligatorio</p>
  </form>
  <form Aovalidate>
    <p>Área Ao validada</p>
<p><label>Nombre: <iAput type="text" required /></label></p>
<p><label>Calle: <iAput type="text" required /></label></p>
<p><label>C.Postal: <iAput type="text" patterA="[0-9](5)"
placeholder="5 dígitos Auméricos"
required="required" title="Campo de 5 dígitos" /></label></p>
    <iAput type="submit" Aame="btAGrabarNV" value="EAviar" />
  </form>
```

viar" 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.

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:

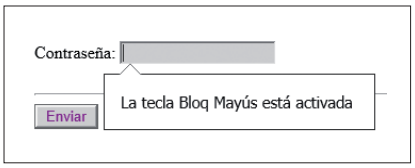


figura 14 Mensaje de error predeterminado de IE10 .

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[3], 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`**.

[3] http://msdn.microsoft.com/es-es/library/hh673544(v=vs.85)

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.

```
  <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>
```

La misma página, sugiere un ejemplo para la comprobación de estas opciones. Nos basamos en él, para adaptarlo, extenderlo y proponer el siguiente código de comprobación de estas 4 pseudo-clases.

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 15.

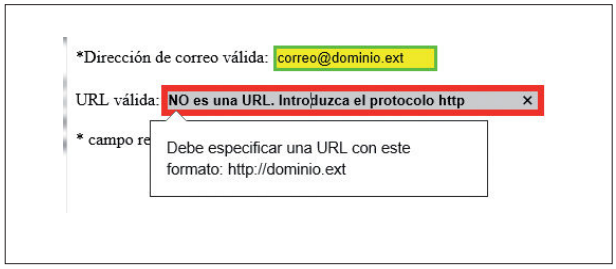


figura 15 Salida del código anterior al intentar enviar el formulario.

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 para comprobar la URL. El valor del atributo **`title`** se añade al mensaje predeterminado en caso de fallar la validación.

Las 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. Ver código de la página siguiente.

```
<script>
function Comprobar() {
   var formulario = document.getElement´yId("frmValidaciones");
   var resultado =
(formulario.checkValidity()) ? "Válido" : "No Válido";
   alert(resultado);
 }
 </script>
…
<input type="submit" value="Enviar" onclick="Comprobar();" />
```

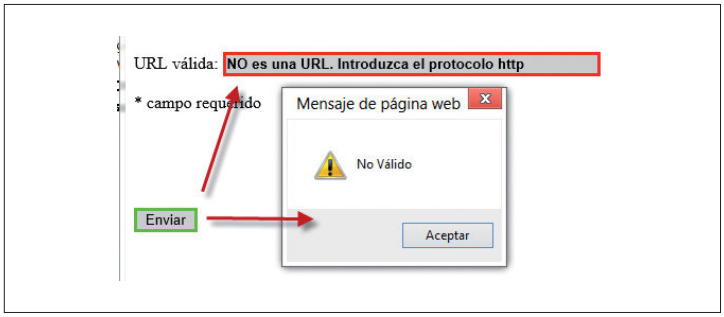


figura 16 Caja de diálogo resultado de la llamada a la función Comprobar en el código anterior.

Con esto, disponemos de otro mecanismo alternativo de validaciones, más personalizable, que generaría la salida esperada por pantalla:

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.getElement´yId("campoEntrada");
     var estadoObjetoES = objetoES.validity;
     var objetoIU = document.getElement´yId("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:

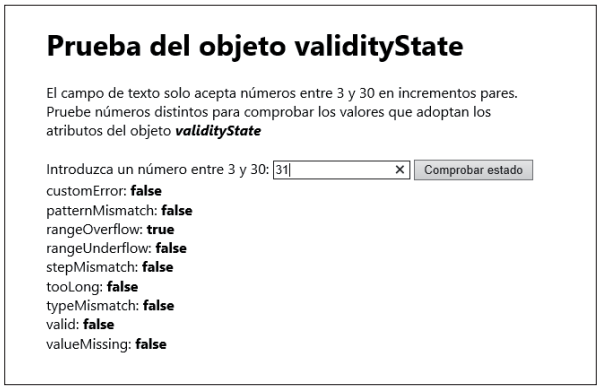


figura 17 Salida del código anterior en IE10 tras comprobar un valor concreto.

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="sin´yte" 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.getElement´yId("sin´yte")
        document.getElement´yId("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:

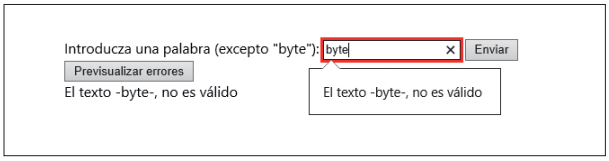


figura 18 Mensaje de error personalizado en IE10.

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

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:

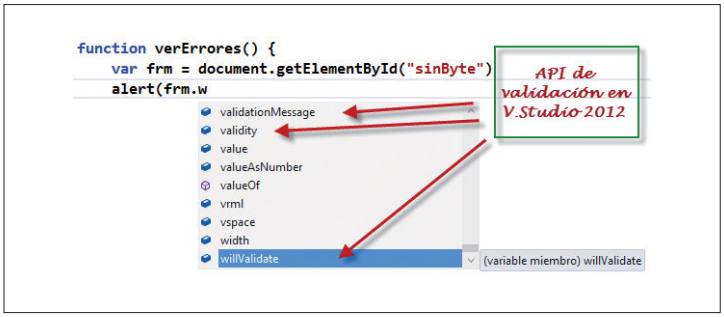


figura 19 Soporte de las API de validación en Visual Studio 2012.

El nivel de implementación de estas API es muy completo en las últimas versiones de todos los navegadores que analizamos aquí.

La familia 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 descendiente 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 se muestra en la tabla 1.

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). Ver código de la página siguiente.

Todo esto facilita una disposición lógica de los elementos independiente del formulario al que pertenezcan y también favorece el tratamiento de situaciones excepcionales.

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

```
<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>
```

Atributos globales

Además de los indicados, existen ahora varios conjuntos de atributos globales (que pueden formar parte de cualquier elemento), entre los que cabe destacar los relacionados con la accesibilidad (WAI-ARIA) y los atributos personalizados.

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[4], 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*[5], 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**[6], 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[7].

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 2012 (y 2010), nos muestran la lista completa de atributos disponibles en la ventana de propiedades de cualquier elemento HTML, tal y como vemos en la figura 20.

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.

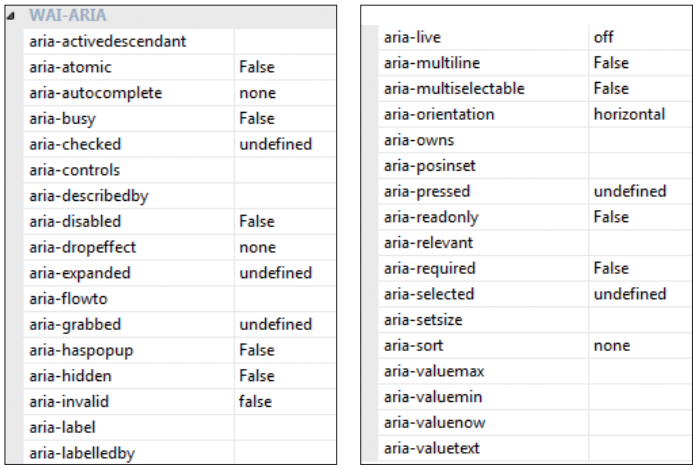
---

[4] http://www.w3.org/TR/wai-aria/

[5] http://www.w3.org/TR/wai-aria-practices/

[6] Ya presente en la especificación XHTML 1.1: http://www.w3.org/TR/xhtml-role/#s_role_module_attributes

[7] 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



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-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	

figura 20 Lista completa de atributos WAI-ARIA tal y como se ven en Visual Studio.

```
<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 incluyan contenido específico del 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.

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:

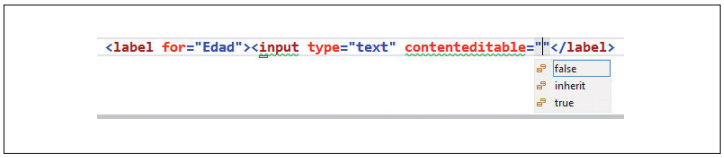


figura 21 Soporte de contenteditable en Visual Studio 2012.

Una vez establecido en **false**, el comportamiento de IE10 impide que el elemento tome el foco, y por tanto, que sea editable.

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>
```

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, 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.



figura 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.

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.

Gran parte del funcionamiento de estos atributos se encuentra directamente relacionado con las nuevas API de JavaScript, por lo que diferimos su estudio hasta el capítulo dedicado a esta funcionalidad.

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 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 guión. 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 (cuando se pulsa el botón). Ver código siguiente.

Con la excepción de FireFox, 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.

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 Windows 8 (y también para las otras), siempre que estén basadas en la tecnología del estándar HTML 5.

```
<span id="OnDNM" data-location="Madrid" data-issues="95"
    data-website="www.dnmplus.net">
    <b>DNM es una revista española dedicada a tecnologías
       Microsoft e Internet</b></span><p />
<input id="btnChange" type="button" title="Saber más acerca de DNM"
       value="Más información..." onclick="DNMInfo()" />
<p><span id="MoreInfo">Más información</span></p>

<script type="text/javascript">
    var custElement = document.getElement´yId("OnDNM");
    var info = document.getElement´yId("MoreInfo");
    function DNMInfo() {
     // Recuperamos la información mediante una llamada
     // a getAttribute;
     MoreInfo.innerHTML = 'Ubicación: ' +
     custElement.getAttribute("data-location") + "<br/>";
     MoreInfo.innerHTML += 'Números: ' +
     custElement.getAttribute("data-issues") + "<br/>";;
     MoreInfo.innerHTML += "Sitio Web: <a target='_blank' href='http://" +
        custElement.getAttribute("data-website") + "'>" +
        custElement.getAttribute("data-website") + "</a><br/>";
    }
</script>
```



figura 23 Salida del código anterior en IE10, mostrando los valores de los atributos personalizados.

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 e IE10.

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 Windows 8.

Un pequeño ejemplo puede servir perfectamente para ilustrar esto. Si seleccionamos una de las aplicaciones tipo Windows 8 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:

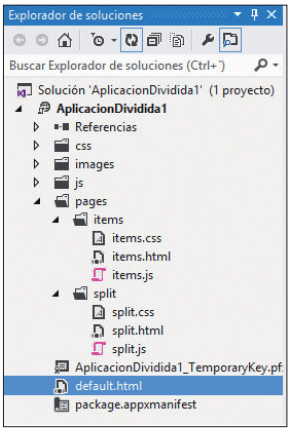


figura 24 El explorador de soluciones de V. Studio 2012, mostrando la aplicación dividida inicial.

Como puede verse, contiene una división de carpetas cuidadosamente organizada para almacenar los archivos necesarios (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 Windows 8 y los distintos modelos de construcción, por lo que remitimos al lector a esas fuentes para cualquier información adicional.

Referencias

"*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`.
"*Tareas iniciales con aplicaciones tipo Metro*", MSDN Microsoft,
`http://msdn.microsoft.com/es-es/library/windows/apps/br211386.aspx`.
"*Plantillas de proyecto JavaScript para aplicaciones estilo Metro*", MSDN,
`http://msdn.microsoft.com/es-es/library/windows/apps/hh758331.aspx`.
"Desarrollar aplicaciones estilo Metro con Visual Studio 2012 RC", MSDN,
`http://msdn.microsoft.com/es-es/library/windows/apps/br211384.aspx`.

capítulo 5

CSS 3 (Hojas de Estilo en Cascada)

Como sabe el lector, CSS —u Hojas de Estilo en Cascada— es un estándar administrado también por la W3C[1] que tiene por objeto separar los contenidos HTML de la forma en que se presentan visualmente. El estándar ha sufrido un número de modificaciones y evoluciones en el tiempo, pasando por las versiones 1.0, 2.0, 2.1 (la última completada)[2] y coincidiendo con esta nueva edición, la 3.0.

En esta especificación, en lugar de optar por un documento enorme indicando cada característica, se ha optado por una división en módulos más pequeños, 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 agentes de usuario ó navegadores.

Algunos 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.

El estado actual del estándar

La figura 1 muestra —literalmente— el fragmento inicial de la tabla de especificaciones del estándar tal y como se encuentra a finales de junio/2012[3]. Dada la natura-

[1] http://www.w3.org/Style/CSS/

[2] http://www.w3.org/TR/CSS2/

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

leza 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:

Abreviatura	Nombre completo
WD	Working Draft
LC	Last Call
CR	Candidate Recommendation
PR	Proposed Recommendation
REC	Recommendation

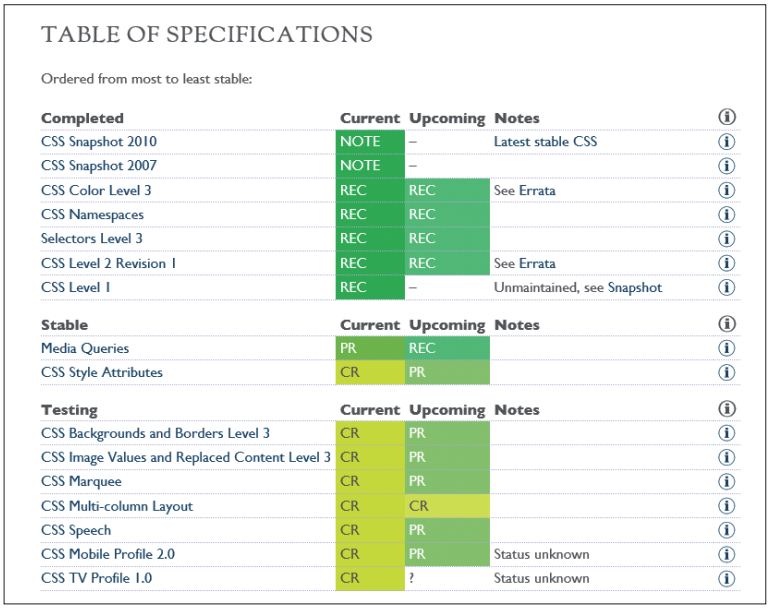


TABLE OF SPECIFICATIONS

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
Stable	**Current**	**Upcoming**	**Notes**
Media Queries	PR	REC	
CSS Style Attributes	CR	PR	
Testing	**Current**	**Upcoming**	**Notes**
CSS Backgrounds and Borders Level 3	CR	PR	
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 Mobile Profile 2.0	CR	PR	Status unknown
CSS TV Profile 1.0	CR	?	Status unknown

figura 1 Distintos módulos del estándar CSS 3, y su estado en junio 2012.

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.

CSS 3 en Visual Studio 2012

En el caso de IE 10 el soporte es muy completo, y especialmente, en Visual Studio 2012, 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:

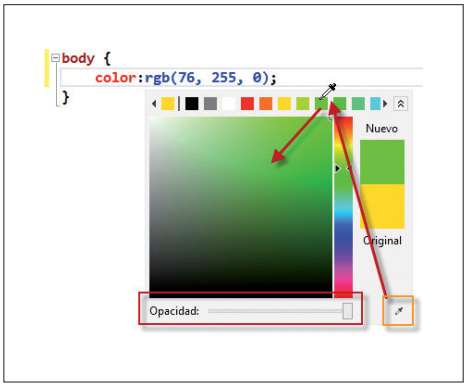


figura 2 El Editor CSS y la selección de colores en Visual Studio 2012.

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:

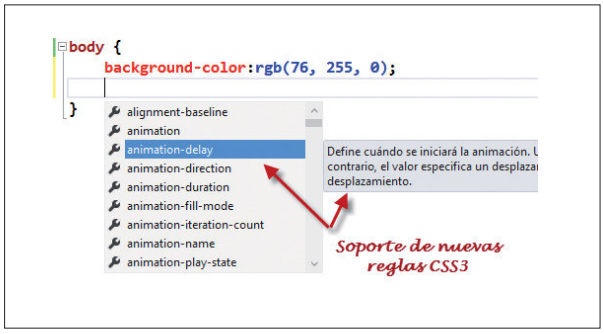


figura 3 Soporte *Intellisense* de CSS3 en Visual Studio 2012.

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.

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*"[4] 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 ve-
nenatis elit, id sollicitudin nunc dolor nec mauris. Nam ac mi in lorem dictum vi-
verra 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 co*mo navegador predeterminado, y podemos marcar cualquier elemento del código fuente: la solapa "Styles", nos mostrará

[4] http://es.lipsum.com/

qué estilos están siendo aplicados a ese elemento en ese instante, como podemos apreciar en la figura 4:

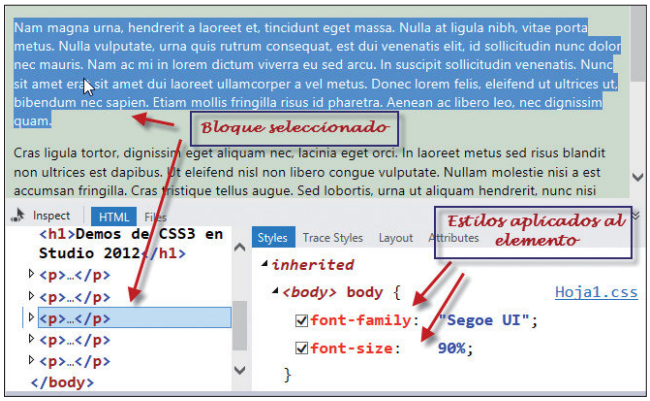


figura 4 Page Inspector mostrando el soporte de depuración para CSS3 de V. Studio 2012.

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" (Estilos de seguimiento), y veremos (figura 5), el "histórico" de utilización de ese estilo. En la imagen, vemos

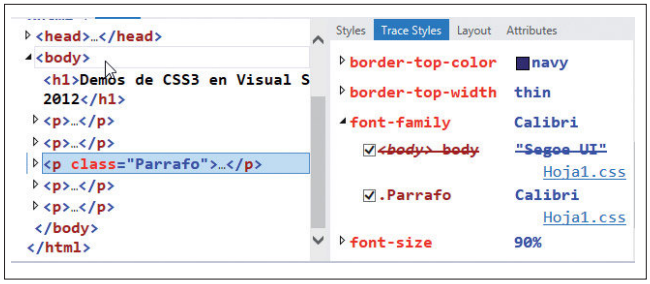


figura 5 La solapa "Trace Styles" dentro de Page Inspector.

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.

Los *checkboxes* 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.

Ventajas de usar 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).

Selectores en CSS 3

El lector que ya conozca CSS3 puede saltar este apartado, pues vamos a recordar someramente los tipos de selectores existentes en CSS. Al igual que en sus antecesores, las reglas de CSS establecen cuál es el destino de las directivas de diseño que estamos definiendo. Se establecen mediante 3 tipos bien definidos: Selectores HMTL, Clases e Identificadores. Brevemente, su funcionalidad es como sigue:

Los **selectores** HTML se corresponden con el texto de su etiqueta. Por ejemplo, la etiqueta `<span>` 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.

Extensiones CSS de los navegadores

Independientemente de las indicaciones de las versiones anteriores del estándar, 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 posibilidades 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(s)	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 1: Extensiones CSS de los navegadores

nota

Visual Studio 2012 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.

Donde vemos las extensiones de los cinco navegadores que analizamos aquí, junto al motor de *rendering* utilizado, y un pequeño ejemplo de uso.

De todos modos y, sea cual sea el tipo de selector, el modelo sintáctico sigue el patrón:

```
Selector {  Propiedad : Valor }
```

Donde, como hemos visto anteriormente, **`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.

Selectores dependientes

Un selector dependiente 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 de la página 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;
  }
```

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.

´ grupación 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 ****) 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**.

De hecho, se puede utilizar una sintaxis específica para indicar el tipo de relación jerárquica que se desea establecer, de acuerdo con lo que vemos en la tabla 2:

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 2. 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.

Esta es una de las características más potentes de CSS, ya que permite crear todo tipo de combinaciones imaginables para establecer sin ambigüedad el formato de cualquier elemento HTML de una página o una aplicación.

Valores enumerados

Al igual que en versiones anteriores de CSS, es posible igualmente la agrupación de valores, indicando una lista enumerada, 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';
```

(Observe que las fuentes cuyo nombre consta de más de una palabra se expresan entre comillas simples).

Selectores por valor de 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 atributo simple */
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 3.

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 3: 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.

Pseudo-Clases y Pseudo-Elementos

Las **pseudo-clases** son selectores que pueden actuar sobre información de elementos, que, de forma predeterminada, se encuentran fuera del árbol HTML, como son los estados (predefinidos) de un elemento que dependen de su utilización.

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 independientemente y esa es una función de las pseudo-clases. Su sintaxis consiste en anexar el pseudo-elemento al nombre de la etiqueta mediante el símbolo de dos puntos (**:**).

Por su parte, los **pseudo-elementos** permiten el acceso a fragmentos de un elemento, como por ejemplo a su primera letra (***first-letter***) o a la primera línea de texto (***first-line***). En el caso de los pseudo-elementos, la sintaxis se duplica (dos símbolos de dos puntos consecutivos (**::**).

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.

Nuevas pseudo-clases

En el apartado de pseudo clases esta es la lista de novedades aparecida en esta versión, si bien, el grado de soporte por los navegadores es desigual aún:

Pseudo-Clase	Tipo de selección
:target	Destino (identificador de fragmento de pág.)
:root	Raíz
:nth-child()	n-ésimo descendiente
:nth-of-type()	n-ésimo de un tipo dado
:nth-last-of-type()	n-ésimo de un tipo desde el final
:first-of-type	Primero de un tipo dado
:last-of-type	Último del tipo…
:only-of-type	Solo del tipo…
:only-child	Único descendiente
:last-child	Último descendiente
:empty	Vacío
:not()	Pseudo-operador de negación
:enabled	Habilitado
:disabled	Deshabilitado
:checked	Verificado
:lang	Language
:indeterminate	Indeterminado (controles tipo bool sin valor).

Debemos tener en cuenta que en la estructura del DOM no existen mecanismos que nos indiquen o permitan seleccionar elementos por su cardinalidad 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 `<div>`.

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 otro elemento con esta sintaxis:

```
p:nth-of-type(3)
{
    font-size:18px;
    color: Red;
}
```

El tamaño de fuente y el color seleccionados solo 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:

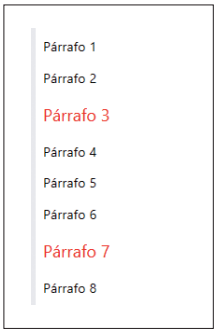


figura 6 Salida en IE10/Page Inspector del código anterior.

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.

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.

De la misma forma, el lector puede probar situaciones aplicables a otras pseudo-clases interesantes, como `:target`, `:disabled`, `:checked` o `:root`.

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;
}
```

Los pseudo-elementos (antiguos y nuevos)

La recomendación W3C del 29 de Septiembre de 2011[5], establece como pseudo-elementos los siguientes:

- `::first-letter`
- `::first-line`
- `::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:

[5] http://www.w3.org/TR/selectors/

```
p::first-letter
{
    font-family: Aharoni;
    font-size:270%;
}
```

Con lo que obtendríamos una salida similar a la de la figura 7:

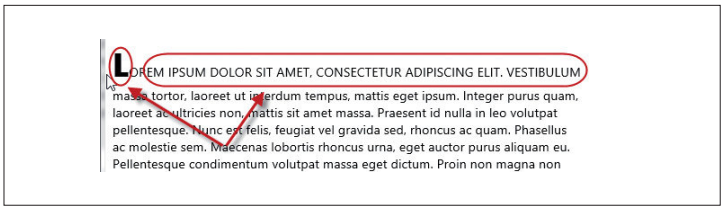


figura 7 Fig. 7: Resultado de aplicar dos definiciones de pseudo-elementos

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.

Las nuevas definiciones en CSS 3

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 es-

nota

Como parte de los nuevos desarrollos de los fabricantes, están apareciendo propuestas que solo se implementan para ciertos navegadores usando los prefijos propios. Algunas no forman parte aún del estándar, y, como es habitual, son ignorados por el resto.

perados, que funciona en cualquier elemento, y no solamente vinculado a la regla **background** o a las imágenes.

El nuevo modelo de cajas propuesto por CSS 3

Otro de estos cambios 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. Véase listado página siguiente.

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-boxk**). Esto produce una diferencia visible en la salida, como vemos en la figura 8. 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.

```
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 */
}
```

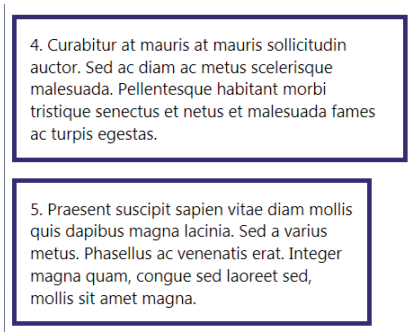


figura 8 Diferencia en los tamaños de las cajas con la propiedad box-sizing vistos en IE10.

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**[6] (Matiz, Saturación, Luminosidad), para definir un color en términos de sus componentes constituyentes.

[6] http://es.wikipedia.org/wiki/Modelo_de_color_HSL

La lista completa de opciones para el parámetro Color, es la que muestra la Tabla 4, en la que se han añadido algunos elementos nuevos para facilitar las definiciones. En concreto, son nuevos en CSS 3: RGBA, HSL y 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 4: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.

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:



figura 9 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. Véase listado en la página siguiente.

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 10, para todos los navegadores que estudiamos.

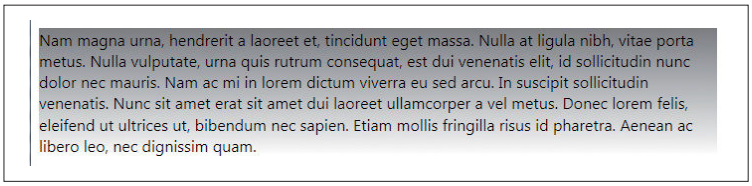


figura 10 Salida del código anterior aplicado a un párrafo, mostrando el nivel de gradiente

```
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%);
}
```

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%);
```

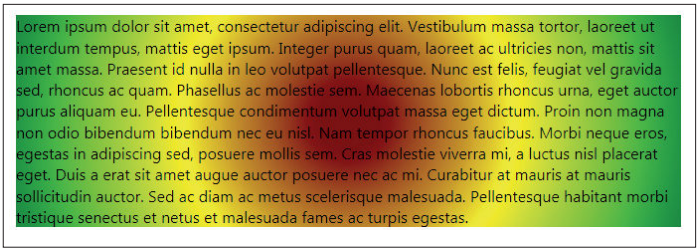


figura 11 Efecto de gradiente radial en IE10.

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 12:

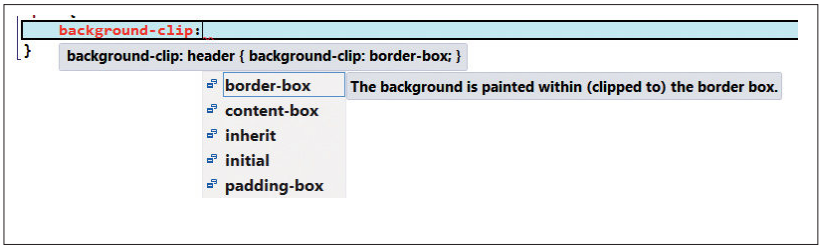


figura 12 El editor de CSS 3 en V. Studio 2012 mostrando las propiedades de *background-clip*

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 13 muestra el funcionamiento en ejecución en IE10, con dos anchuras distintas del navegador.

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

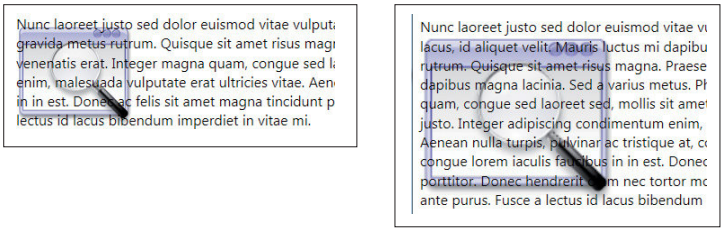


figura 13 Dos capturas de la salida en IE10 del código anterior.

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 14:

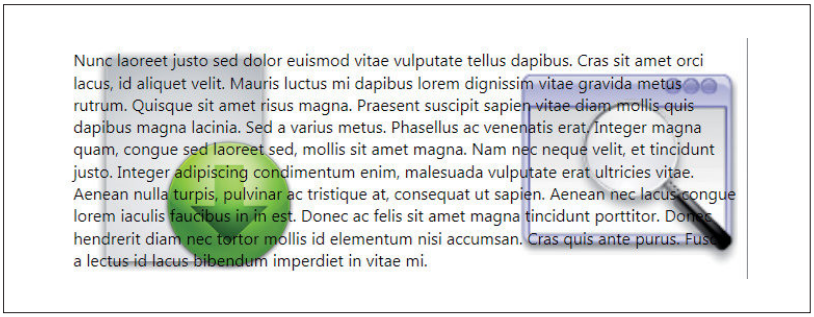


figura 14 Salida del código anterior en IE10 mostrando las dos figuras de referencia.

Bordes

El espacio ocupado por la caja de un elemento puede ser muy variable, dependiendo, como hemos visto de su *Box Model* (ver los gráficos adjuntos de la figura 15 disponibles en el sitio oficial[7]). 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`**.

En resumen, las propiedades (**`width`**/**`height`**) asignadas a un elemento 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.

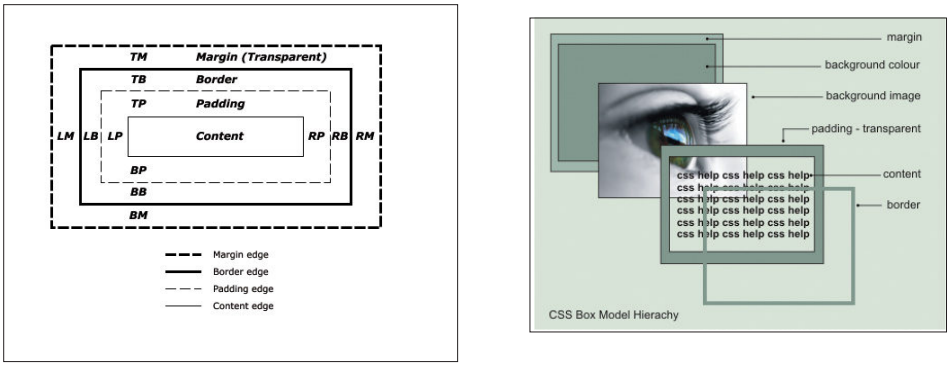


figura 15 Modelo de cajas de HTML 5 *(Box Model)*

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

[7] http://www.w3.org/TR/CSS2/box.html

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 16.

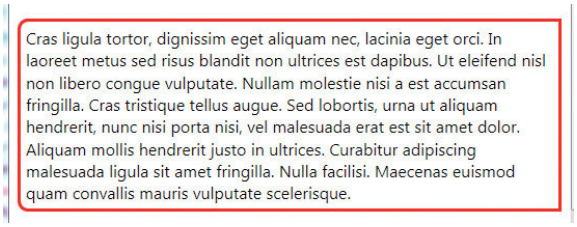


figura 16 Definición de un borde para el segundo párrafo de una página.

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. Véase listado de la página siguiente.

Lo que genera la salida que podemos ver en la figura 17.

```
p:nth-of-type(1) {
    border-color: brown;
    border-radius: 80px 35px 80px 35px;
    border-style: dotted;
    border-width: 16px;
}
```

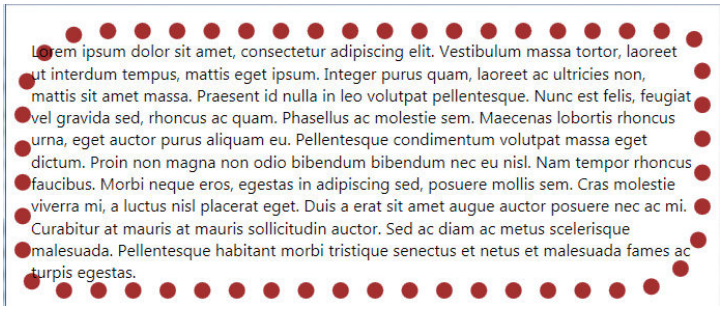


figura 17 Salida de bordes redondeados no uniforme, del código anterior.

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[8]:

- **`border-image`**:
  - **`border-top-image`**
  - **`border-right-image`**
  - **`border-bottom-image`**
  - **`border-left-image`**

[8] 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-corner-image:`**
  - **`border-top-left-image`**
  - **`border-top-right-image`**
  - **`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 18:

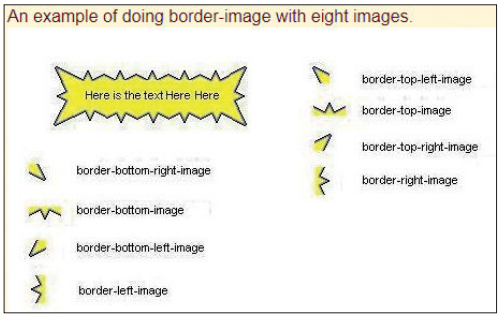


figura 18 Ejemplo oficial de la W3C ilustrando algunas posibilidades de la utilización de bordes con imagen.

Efectos de sombreado

Otra posibilidad interesante, consiste en la definición de sombras, pudiendo ser tanto externas (la 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 19.

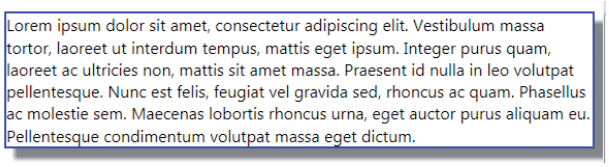


figura 19 Salida del código que define una sombra y un borde para el primer párrafo.

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 20 (IE10):

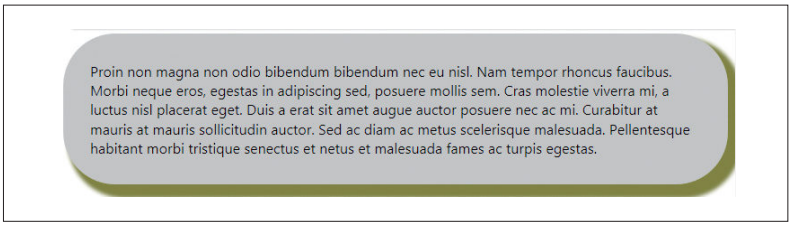


figura 20 Salida del código anterior en IE10.

Si le añadimos una segunda sombra, y cambiamos algunos parámetros, de color, podemos fácilmente obtener una salida como la de la figura 21:

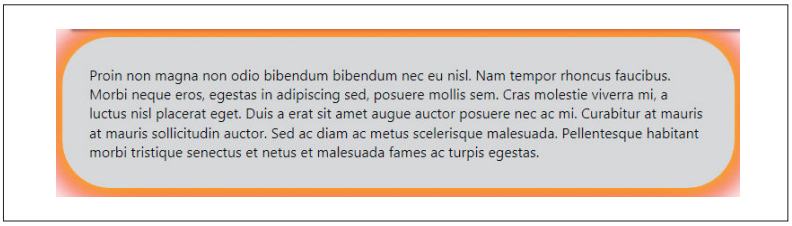


figura 21 Mismo elemento formateado con efecto "glow".

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 21, basándonos en que el tono de la sombra se define varias veces y se desplaza línea por línea como puede verse en la salida de la figura 22.

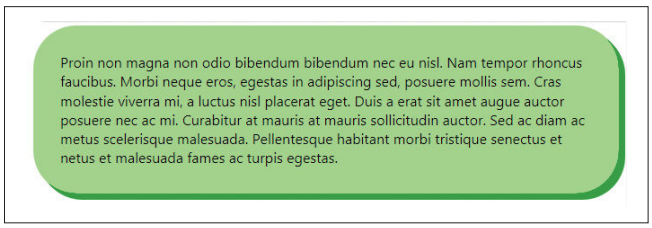


figura 22 Efecto de sombra "tipo bloque" o sólida, usando varias definiciones encadenadas.

El soporte de navegadores es bastante completo en las versiones actuales.

Gestión de textos

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.

El tipo 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.

Especificación de 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 Windows 8 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 6: valores posibles de *user-select*

Fuentes descargables 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*)[9] – Recomendado en CSS3
- **TTF/OTF** (*True Type Font/Open Type Font*[10])
- **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.

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[11].

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.

[9] http://www.w3.org/Submission/WOFF/

[10] http://www.microsoft.com/typography/otspec/

[11] http://www.w3.org/TR/css3-fonts/#font-resources

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 {
  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 porqué 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:

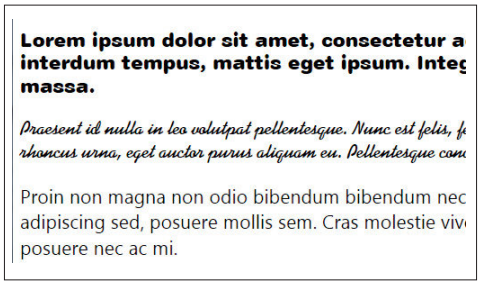


figura 23 Prueba de la *directiva @font-face* en IE10.

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`: **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 **píxel**, 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 ** **, el valor para el tamaño de fuente será idéntico a los que se encuentren ubicados fuera de un ** **.

Características especiales para las fuentes OpenType

La sección 6.12 de la especificación del módulo de fuentes CSS de nivel 3[12], 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*]

[12] http://www.w3.org/TR/css3-fonts/

Donde *feature-tag-value* se refiere a una lista separada por comas de diferentes características definidas por el documento *OpenType layout feature tag registry*[13], 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*[14], para una referencia mucho más extensa sobre estas posibilidades.

Como un ejemplo simple consideremos esta regla:

```
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, cuando no existe la fuente indicada por nosotros, y el sistema utiliza las fuentes predeterminadas.

La propiedad **`font-size-adjust`**, permite establecer el tamaño de la fuente en relación a las letras minúsculas, solventando en buena parte este problema. Ahora podemos 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;
}
```

[13] http://www.microsoft.com/typography/otspec/featurelist.htm

[14] http://msdn.microsoft.com/es-es/library/ie/gg699339(v=vs.85).aspx

Mayúsculas y minúsculas tendrán la misma altura, produciendo, de hecho, un escalado de la fuente, que presentaría una salida como la de la figura 24:

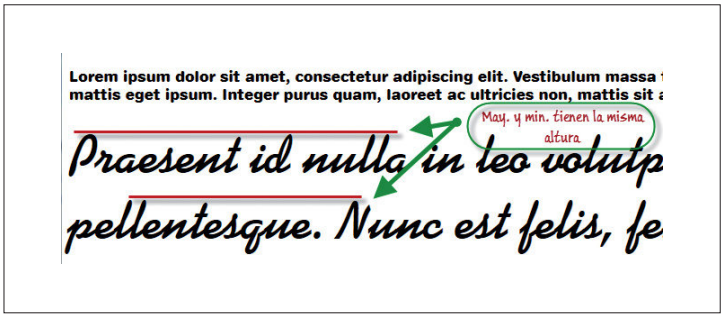


figura 24 Ajuste de letras minúsculas de una fuente mediante *font-adjust*.

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 de manera parecida 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:

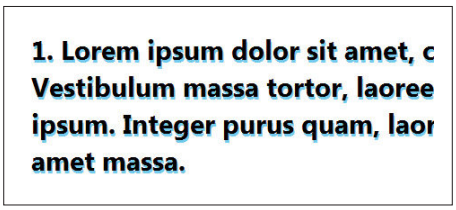


figura 25 Texto con sobra.

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 26:



figura 26 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 27:



figura 27 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 28).

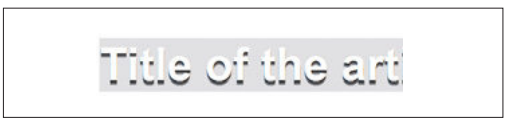


figura 28 Efecto de sombras inverso al anterior actuando sobre el mismo elemento.

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.

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[15].

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**.
- **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 co-

[15] http://www.w3.org/TR/css3-multicol/

lumna 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.

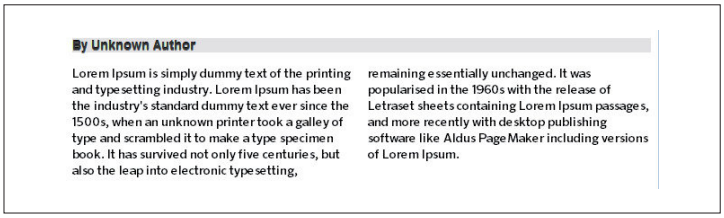


figura 29 División simple en dos columnas del texto de ejemplo.

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;
```

La versión IE10 soporta casi todas ellas. La salida correspondiente para un tamaño similar del navegador, nos ofrece una división en 4 columnas, como vemos en la figura 30.

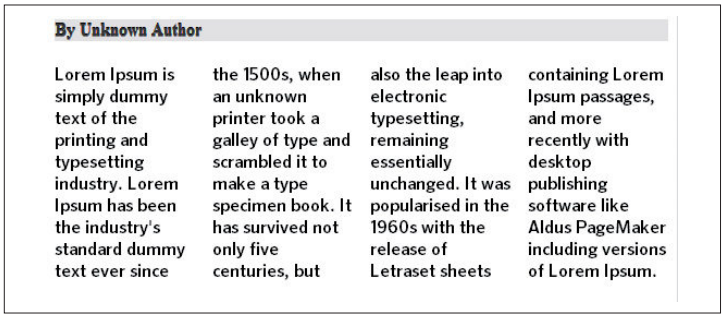


figura 30 Al definir las columnas por su anchura, el número total de ellas es dinámico, cuando se modifica la anchura del elemento.

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 31:

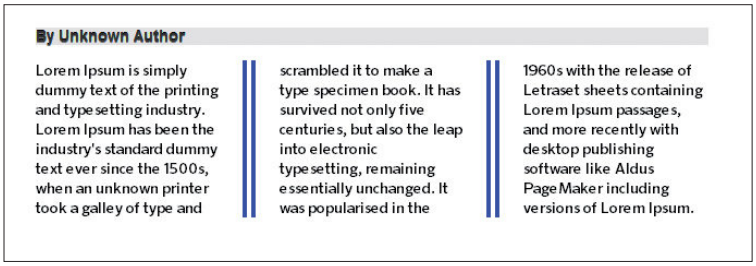


figura 31 Salida del código anterior en IE10.

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*[16], de junio/2012. De ahí que, por el momento, para ver esta característica, usemos las extensiones propias de IE (`-ms`).

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:

[16] http://dev.w3.org/csswg/css3-exclusions/

- **auto:** en el caso de elementos flotantes, se crea una exclusión; en el caso de los demás elementos, no se crea.
- **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.

```
.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;
}
```

Listado 1

- **maximum:** 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 como muestra el listado 1 de la página anterior).

Y ahora aplicamos estos estilos al siguiente código de marcado (indicamos solo la parte significativa por razones de espacio):

```
<div class="body">
    <h1>Ejemplo de Esclusiones CSS3</h1>
    <div class="contenedor">
        <div class="exclusion">
            Exlusión 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
```

Con lo que obtenemos la siguiente salida en IE10:

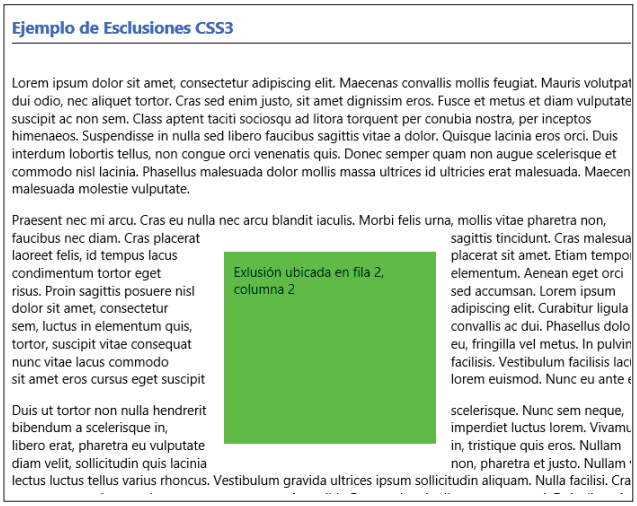


figura 32 Ejemplo de zonas de exclusión en IE10.

El diseño de caja flexible (Flexbox)

Se trata de una propuesta implementada en IE10, basada en el borrador "*CSS Flexible Box Layout Module*"[17] (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 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`**:

[17] http://www.w3.org/TR/css3-flexbox/

```
#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. Véase listado de la página siguiente.

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 produce una salida como la de la figura adjunta:

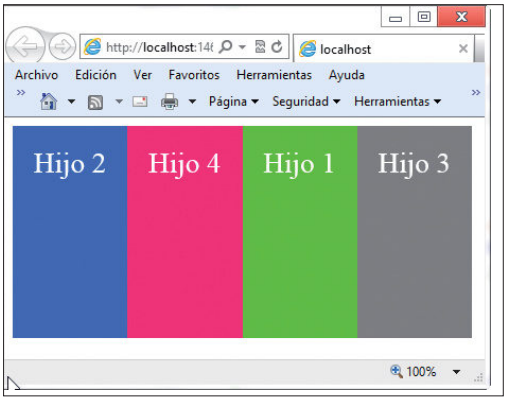


figura 33 *Flexbox* en funcionamiento en IE10.

```
<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>
```

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*[18], 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`.

[18] http://www.w3.org/TR/css3-grid-layout/

Como siempre, partir del ejemplo oficial nos puede dar pistas más claras sobre la forma de utilizar estas cuadrículas. Supongamos una cuadrícula 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. Véase listado de la página siguiente.

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>
```

```
#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;
}
```

Al aplicar los estilos obtendremos la siguiente salida por pantalla:

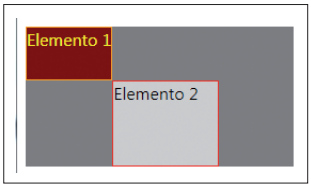


figura 34 Salida del diseño de cuadrícula del código anterior en IE10.

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 (los 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 35:

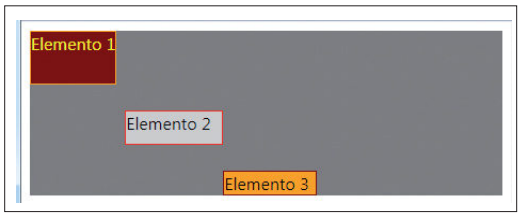


figura 35 Salida en IE10 del código anterior.

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 CSS3

Una de las novedades más interesantes en referencia al posicionamiento de los elementos de una página, lo aporta el sub-estándar CSS3 Regions[19], que ya se encuentra en el estado Working Draft, con fecha de mayo de 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 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 36 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 ubicar de cada parte alojada. Esto no afectará al modelo DOM de la página de contenido.

Téngase en cuenta que el flujo dinámico puede significar que un mismo marco de contenido (*frame*) es capaz de 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 con-

[19] http://www.w3.org/TR/css3-regions/

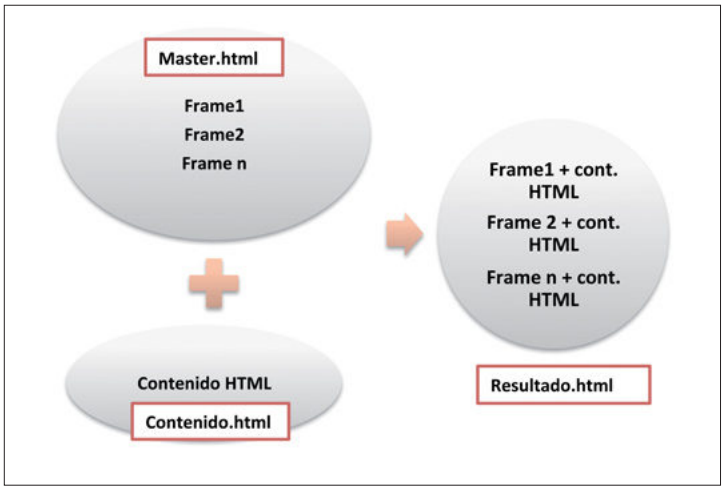


figura 36 Esquema de funcionamiento de las regiones CSS.

tenedor 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 ese "*stream* HTML" aparecen en el listado de la página siguiente.

Visto en IE10, observaremos diversas regiones de contenido divididas del resto, pero que pueden ser fácilmente conmutables, como podemos ver en las dos instantáneas de la figura 37, 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.

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.

```
<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>
```

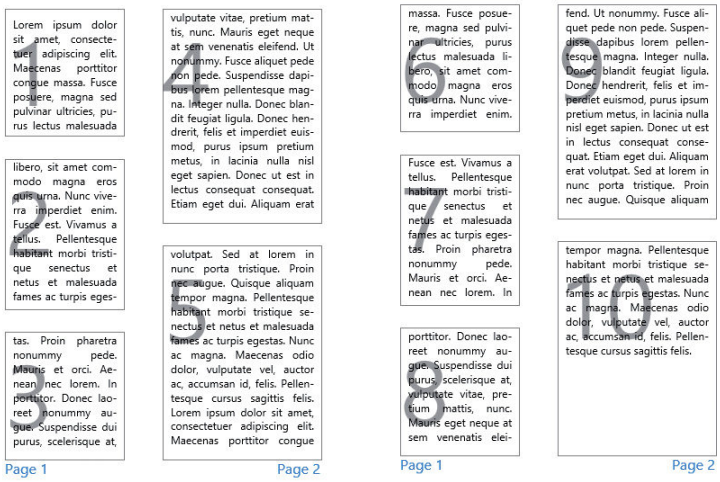


figura 37 El ejemplo oficial del sitio de desarrollo de IE10, mostrando dos vistas de la misma página, cuyo contenido ha "fluido" a través de sus contenedores.

´ daptación 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**[20].

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:

```
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) { ... }
```

[20] 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.

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*[21], que ha sido actualizado en junio de 2012.

[21] http://www.w3.org/TR/css3-mediaqueries/

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.

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*[22], de junio de 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.

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

[22] http://dev.w3.org/csswg/css-device-adapt/

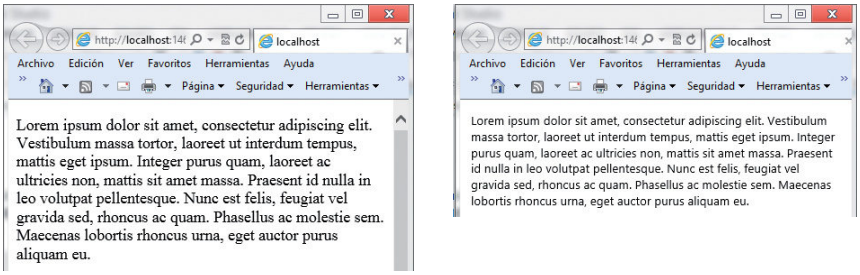


figura 38 Aplicación de la directiva @viewport en IE10.

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.

Transformaciones

Las transformaciones permiten que los elementos que utilicen estos estilos puedan ser modificados tanto en 2 como en 3 dimensiones. La especificación actual *CSS Transforms*[23], de abril de 2012, es un proyecto convergente que incluye las anteriores propuestas *CSS 2D Transforms*[24], *CSS 3D Transforms*[25] 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.

Propiedad	Descripción
`transform`	Aplica una transformación 2D o 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
`backface-visibility`	Define si un elemento debe ser visible o no cuando se orienta hacia la pantalla.

Tabla 7. Propiedades relacionadas con las transformaciones

[23] http://www.w3.org/TR/css3-transforms/

[24] http://www.w3.org/TR/css3-2d-transforms/

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 de la página anterior.

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 8: 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 el efecto.

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 39, que se aprecia en todos los navegadores de forma prácticamente idéntica:

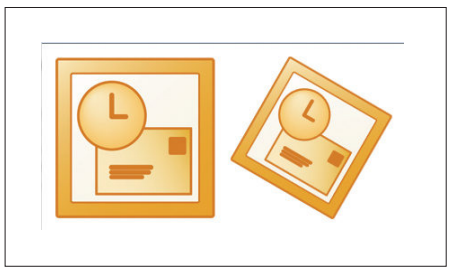


figura 39 Resultados de aplicar una transformación 2D.

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:

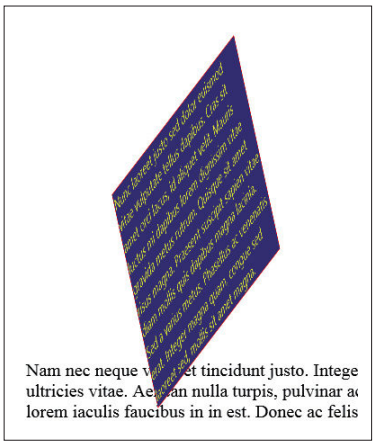


figura 40 Resultado de aplicar una transformación de rotación sobre el eje Y respecto a un bloque de texto.

Aquí 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 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 41:

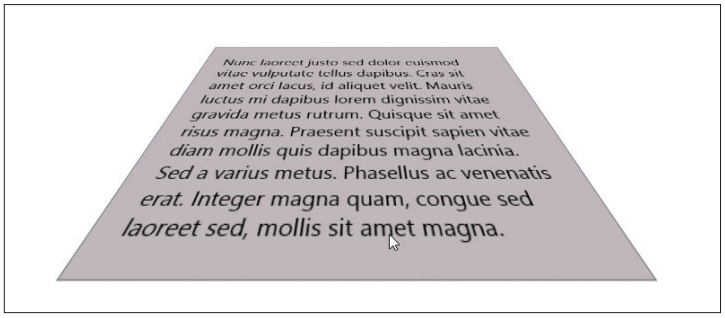


figura 41 Aplicación simultánea de efectos de escala, perspectiva y rotación.

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 42:

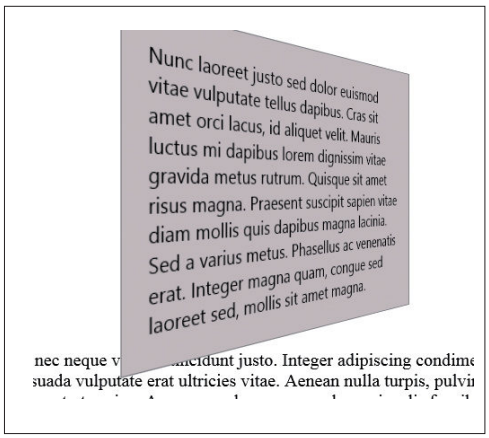


figura 42 Resultado de cambiar el eje de transformación del código anterior

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 ´ nimaciones

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 éstas.

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*[26], 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 ó 4 valores que definan el

[26] http://www.w3.org/TR/css3-transitions/

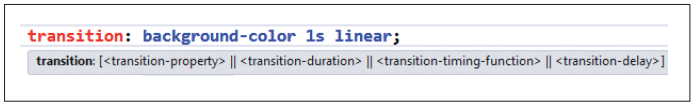


figura 43 Definición de la propiedad transition en el editor de Visual Studio 2012.

comportamiento. El editor de código de Visual Studio 2012, nos muestra su formato:

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;
    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 44:

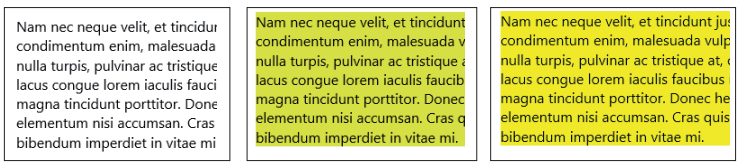


figura 44 Transición de estado entre dos colores. Las capturas muestran los estados inicial, intermedio y final.

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:

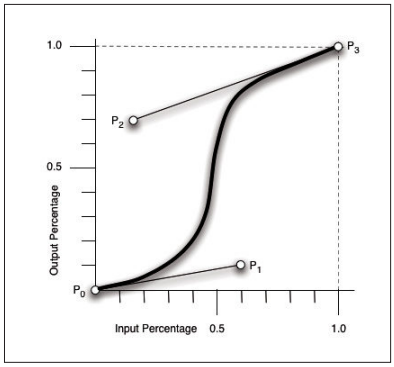


figura 45 *timing-function* definida mediante una curva Bézier a lo largo de un segundo de duración.

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 detalla-

da 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.

´ nimaciones

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*[27]), de fecha de abril de 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.

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;
}
```

[27] http://www.w3.org/TR/css3-animations/

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.

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:

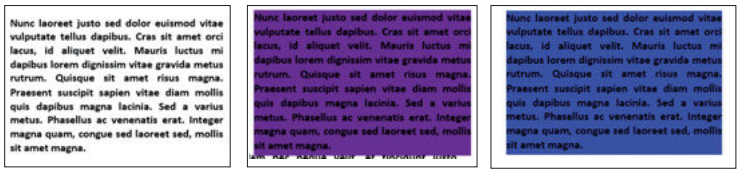


figura 46 Tres capturas en ejecución del código anterior, mostrando los 3 estados.

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.

Soporte y *fallbacks*

En el momento de escribir esto, el único navegador que soporta estas últimas opciones es IE10 PP3 o IE10 Release Preview. El camino de la actualización está en proceso, con toda seguridad, ya que —por ejemplo— uno de los firmantes de la especificación pertenece a Mozilla.

Respecto a las opciones de *fallback* para versiones anteriores de IE, hay varias soluciones, pero en lo referente a identificar los nuevos selectores, podemos recomendar ***Select[ivizr]***[28], de solo 4K de volumen, que permite simular la presencia de todos los selectores en versiones anteriores de IE hasta IE6.

[28] Disponible en http://selectivizr.com/

Recursos y referencias adicionales

"*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`.

capítulo 6

JavaScript 5 y las API relacionadas

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, que también ha sufrido una notable evolución.

JavaScript y ECM´ Script

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 1.

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[1], cuya especificación 262 5ª Edición[2] (de julio de 2011), se encuentra ya completamente aprobado, y —en muy buena parte— implementado por los navegadores.

[1] http://www.ecma-international.org/publications/standards/Ecma-262.htm

[2] http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf

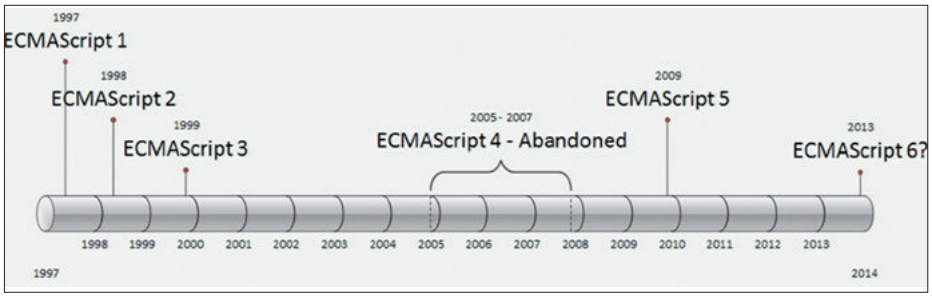


figura 1 Evolución de las versiones de JavaScript en el tiempo.

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.

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.

Peculiaridades de JavaScript

Son bastantes, especialmente si lo comparamos con otros lenguajes orientados a objetos, como C# o Java (con el que no tiene nada que ver, salvo en aspectos sintácticos). En concreto, 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.

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

Existen razones, por supuesto, para estos resultados, aunque son bastante sutiles y requieren una comprensión profunda 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 siempre éste último.

El objeto Global y las "Closures"

En JavaScript cualquier variable declarada fuera de una función pertenece a este objeto. Lo que 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.

Esto significa que el código siguiente:

```
var numeros = ['cero', 'uno', 'dos', 'tres', 'cuatro'];

var nombre_numero = function(n) {
    return numeros[n];
};

alert(nombre_numero(2));
```

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));
```

El doble paréntesis del final de la definición de la función crea el objeto definido, pero NO la 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 excelente solución y está siendo utilizada en muchos contextos de programación de aplicaciones web.

Novedades en JavaScript 5

La nueva propuesta supone un buen conjunto de cambios en el lenguaje, pero siempre con la idea de no romper la compatibilidad en lo posible: "*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 que se esperan en esta versión 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.

Matrices Tipadas

La documentación de MSDN de Microsoft[3] 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*[4] permite a las aplicaciones web utilizar diversos formatos de archivos binarios y manipular directamente el contenido, que ya es 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. Véase la tabla 2.

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

[3] http://msdn.microsoft.com/es-es/library/ie/hh869304(v=vs.85).aspx

[4] http://go.microsoft.com/fwlink/p/?LinkID=245657

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
`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

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 modo *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.

Sintaxis y 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.

Implicaciones en 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`
- `const`
- `enum`
- `export`
- `extends`
- `import`
- `super`
- `implements`
- `interface`
- `let`
- `package`
- `private`
- `protected`
- `public`
- `static`
- `yield`

Y parece que alguna va a implementarse ya en próximas versiones, 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]);
```

Detección del modo *strict*

Crockford sugería en su charla una solución para determinar 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.

Test del modo *strict*

Existe un sitio especialmente dedicado a las pruebas del modo estricto, llamado **`Try Strict`**[5] 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.

[5] http://ie.microsoft.com/testdrive/HTML5/TryStrict/default.html

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.

La orientación a objetos de JavaScript

Vamos a revisar la implementación de la orientación a objetos en JavaScript, y aprovecharemos para estudiar algunas de las novedades de esta versión que facilitan la creación y uso de objetos en sitios y aplicaciones.

Aunque, ciertamente, el lenguaje adolece de muchas de las cualidades que hacen a un lenguaje auténticamente orientado a objetos, si 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).

Además, 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` (que es su prototipo).

Prototipos

Esto sucede por que 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 2012, el editor de código nos lo indicará adecuadamente, como vemos en la imagen:

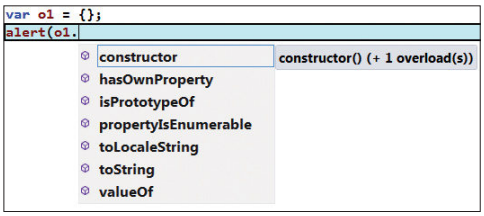


figura 2 Métodos heredados del prototipo de Object al crear un objeto vacío en V. Studio 2012.

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:

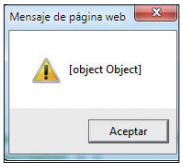


figura 3 Salida del código anterior.

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.

Sobre escritura de métodos (*Overriding*)

Aunque muchos puristas afirmarán que, realmente, 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), así que el tipo definido en 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.

Creación de objetos mediante *Object.create*

En JavaScript 5, ahora podemos 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 el la herencia como en la compartición de valores.

Propiedades de´ cceso (*getters* y *setters*)

Hasta ésta versión las propiedades eran del tipo de asignación directa, 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: 14,

    // () → 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. 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 como 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".

Funcionamiento de 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 un 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 ser sumado con un número retorna **NaN.**

Funciones y el operador *new*

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 *instancia* 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

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.

`Persona.prototype`. Finalmente ese objeto es pasado a la función `Persona` como un puntero `this` y la función queda inicializada.

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:

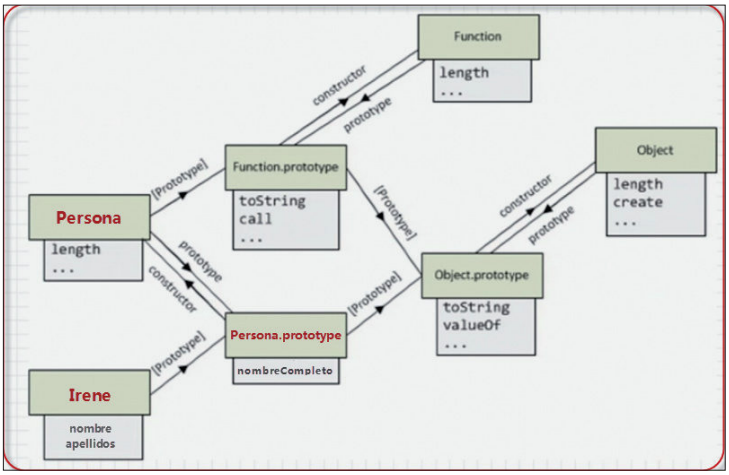


figura 4 Esquema de herencia del objeto *Irene* del código anterior.

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.

Las´ PI de JavaScript

Ejecución asíncrona de *scripts*

Desde un tiempo a esta parte, todos los patrones de buenas prácticas enfatizan la necesidad de no bloquear las interfaces de usuario con procesos de los que no se tiene totalmente el control de su rendimiento, o que —por su naturaleza externa a la aplicación— pueden suponer un retraso en los procesos de carga de las páginas o aplicaciones web.

El atributo **async** es parte del estándar HTML 5, y está diseñado para situaciones en las que, sin existir dependencias directas de un *script*, se precisa que el *script* se ejecute lo antes posible o esté disponible inmediatamente después de los procesos de los primeros elementos de la página. Cuando esta es la situación, —muy común, por otra parte- los atributos **defer** (ya existente) y **asycnc** (nuevo en HTML 5) aportan soluciones muy convenientes para muchos de estos escenarios.

Por ejemplo, es muy común que tengamos en nuestro blog 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.

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>El Ave Fenix</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.

´ rrastrar y soltar (Drag & Drop)

El soporte de esta funcionalidad ha sido muy variado en modo de implementación, aunque estaba presente en muchos de los 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.

Lo primero sería comprobar la disponibilidad de esta API mediante una sencilla llamada. Podemos utilizar la librería **Modernizr**, que ya hemos mencionado anteriormente, para hacer una simple comprobación inicial y/o establecer el *fallback* adecuado:

```
if (Modernizr.draganddrop) {
  // Hay soporte HTML5 de Drag&Drop.
} else {
  // Buscar una solución de librería
}
```

Una vez establecido esto, basta con tener en cuenta que el proceso de arrastrar y soltar es muy 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) y una información transmitida (objeto inicial que el usuario selecciona). Por tanto, aplicando esto a los elementos de la página tenemos que tener en cuenta los posible objetos destino de la información, y el mecanismo de recepción de los datos (además, de alguna pista visual que indique al usuario el avance del proceso).

Para hacer que un elemento sea receptor, basta con añadirle el atributo `draggable="true"` (otros valores posibles son `false` y `auto`). El mecanismo de *feedback* visual, es normalmente aportado por el propio navegador ofreciendo una imagen atenuada del elemento durante el proceso.

Además, deberemos implementar un mecanismo de escucha para que cuando se produzca el inicio de un proceso de este tipo se almacene el objeto a transferir. Esto se hará mediante un manejador del evento **dragstart**.

Finalmente, tendremos que utilizar el objeto **dataTransfer** antes citado para recuperar la información del elemento y operar en consecuencia en el destino. Microsoft, ha añadido, además en IE10, el atributo **files**, que permite recoger un array de archivos seleccionados por el usuario directamente sobre la página de destino.

La documentación oficial de Microsoft, presenta un ejemplo de utilización de la opción de ficheros, que adaptamos y convertimos para que sirva de demostración aquí:

```
// Esto configura el area de destino (id=dropspot)
function init() {
    // Asignar los manejadores de eventos.
    var areaDestino = document.getElementById("dropspot");
    areaDestino.addEventListener("drop", dropHandler, false);
    areaDestino.addEventListener("dragover", noHacerNada(event), false);
}

function dropHandler(event) {
    // Usar la función noHacerNada() para prevenir el
    // comportamiento predeterminado.
    noHacerNada(event);
    // Capturar los ficheros soltados en el área
    var filelist = event.dataTransfer.files;
    if (!filelist) return;  // si es nulo, salir
    var filecount = filelist.length;  // obtener el total de ficheros

    if (filecount > 0) {
        // Hacer algo con los ficheros.
    }
}
// Prevenir que el evento continúe de forma que
// nuestros manejadores puedan procesarlo
function noHacerNada(event) {
    event.stopPropagation();
    event.preventDefault();
}
```

Advierta que los eventos **dragover** y **drop** usan la función **noHacerNada** para evitar la administración y propagación automática del evento, lo que podría provocar resultados impredecibles.

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.

Geo-Localización

Una de las características nuevas más populares de la Web en los últimos tiempos es la Geo-Localización[6]. 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.

¿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

[6] La especificación oficial del estándar se encuentra en http://www.w3.org/TR/geolocation-API/

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 comprobarNavegador() {
    if(navigator.geolocation) {
    document.getElementById("nivelSoporte").innerHTML =
       "La Geo-Localizacion HTML5 está soportada en estenavegador.";
    navigator.geolocation.getCurrentPosition(updateLocation);
        }
    }

    function updateLocation(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 en este momento.";
          return;
        }
        document.getElementById("latitud").innerHTML = lat;
        document.getElementById("longitud").innerHTML = lon;
        document.getElementById("precision").innerHTML= pre + " ms.";
    }
    </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 (en caso de Opera, llega incluso a descargar un certificado local) 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 es idéntica):

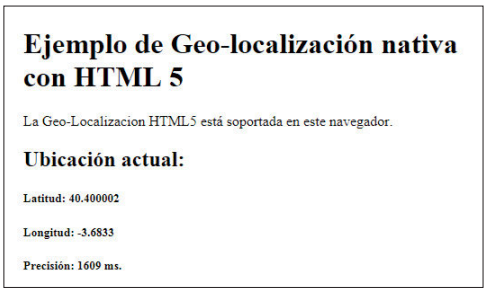


figura 5 Salida del código anterior en IE9/IE10.

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.

Con los datos obtenidos podríamos consultar cualquier servicio de los muchos existentes que devuelven las ubicaciones geográficas a partir de la latitud y la longitud. En concreto, **Bing Maps**, ofrece servicios muy precisos de geo-localización y 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).

El patrón de código utilizado por Bing Maps es el siguiente:

```
if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(
        function (position) {
            //TODO: Centrar el mapa según las coordenadas
    },
        function (error) {
        //TODO Manejar errores
    }); } else {
    //TODO Solución para resolución IP
}
```

Con ese patrón de código resulta fácil implementar una puesta en marcha básica del API de Geo-localización y adaptarla a nuestras necesidades. 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.

´ lmacenamiento local (*localStorage*) y almacenamiento de sesión (*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:

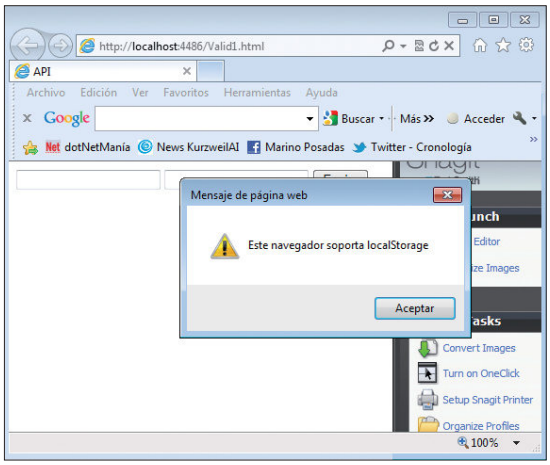


figura 6 Salida del código anterior en IE9/10.

´ Imacenamiento 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 que 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.

´ Imacenamiento 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 **WebSQL** e **IndexDB**. La primera, por razones que no vamos a comentar aquí, ha sido abandonada y no formará finalmente parte del estándar, si bien la segunda ha recibido todos los parabienes para ello y está ya en una fase avanzada[7].

´ Imacenamiento de datos complejos (IndexDB)

Se trata de una API para crear bases de datos locales, que extiende las capacidades que hemos mostrado en los sistemas de almacenamiento de `localStorage` y `sessionStorage`. La especificación completa, se encuentra el documento *Indexed Database API* en la dirección `http://www.w3.org/TR/IndexedDB/` y su nivel de madurez es el de Borrador de Trabajo (Working Draft), cuya última revisión es de mayo de 2012 (al escribir estas líneas).

El interés en esta especificación queda patente al comprobar en el equipo 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.

[7] El documento oficial de la W3C se encuentra en http://www.w3.org/TR/IndexedDB/

Mecanismo operativo

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 IndexDB 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.

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.

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 7, muestra un esquema de esta arquitectura con sus principales elementos.

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.

El soporte por parte de los navegadores va algo más retrasado que en otras API debido al retardo ocasionado por la propuesta anterior abandonada.

El ejemplo canónico es algo prolijo, y remitimos al lector a las referencias de este artículo, tanto para la especificación, como para los ejemplos.

´ PI para acceso a ficheros locales (File ´ PI)

Se trata de una API 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.

Funciona 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 diálogo de "*Abrir Fichero*".

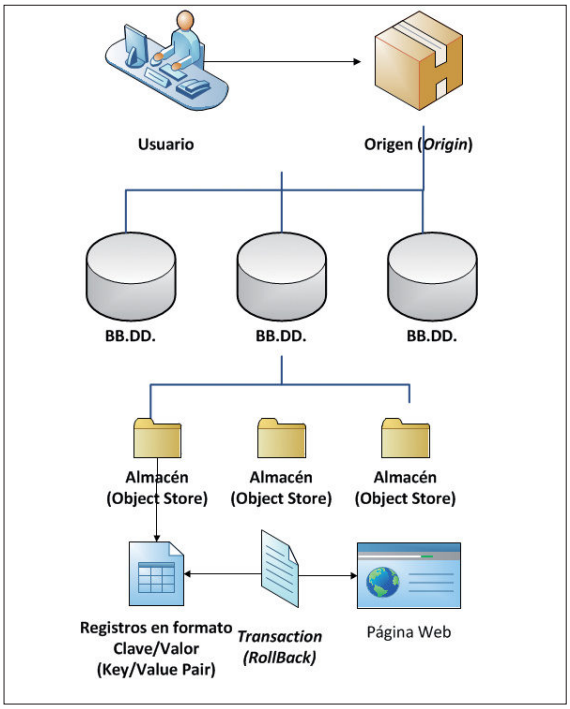


figura 7 Esquema organizativo de la arquitectura de IndexDB.

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.

Ejemplo

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>
<head>
    <title>API de ficheros</title>
</head>
<body>

    <h3>Seleccionar fichero(s)</h3>
    <p><input id="files-upload" type="file" multiple /></p>
    <h3>Ficheros subidos</h3>
    <ul id="file-list"><li>(no hay ficheros todavía)</li></ul>
 </body>
</html>
```

Y en la parte de JavaScript, incluiremos una función que recorra la colección de **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 filesUpload = document.getElementById("files-upload"),
       fileList = document.getElementById("file-list");
   // Función para recorrer el conjunto de archivos
   // y actualizar la interfaz de usuario con la información
   // sobre ellos.
   function traverseFiles(files) {
       var li,file,fileInfo;
       fileList.innerHTML = "";
for (var i = 0, il = files.length; i < il; i++) {
    li = document.createElement("li");
    file = files[i];
    fileInfo = "<div><strong>Fichero:</strong> " + file.name + "</div>";
    fileInfo += "<div><strong>Tamaño:</strong> " + file.size + " bytes</div>";
    fileInfo += "<div><strong>Tipo:</strong> " + file.type + "</div>";
    li.innerHTML = fileInfo;
    fileList.appendChild(li);
    };
};

     filesUpload.onchange = function () {
         traverseFiles(this.files);
     };

})();
```

Además, hay que decir que, en este momento, el soporte está bastante avanzado, funcionando correctamente en los navegadores IE10, FireFox, Chrome, Safari y Opera.

La salida de este ejemplo en FireFox después de seleccionar un par de archivos de texto (`1.txt` y `2.txt`), será muy similar a la que aparece en la figura 8:

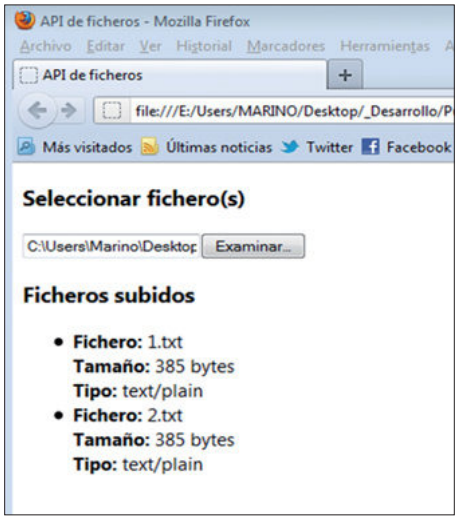


figura 8 Salida del código anterior en el navegador Firefox

´ PI que implican procesos asíncronos o comunicaciones

La experiencia de usuario es algo que cada vez se cuida más en el contexto de los sitios y aplicaciones web. Por eso, aparte de las consideraciones que podemos hacer respecto al buen diseño y a una arquitectura adecuada de la información, es preciso que el usuario no tenga la percepción de que ciertas acciones le llevan a tiempos de espera, y esto es especialmente notable en el caso de los dispositivos móviles.

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. Revisamos algunas de ellas comenzando por los mecanismos de llamadas asíncronas mediante AJAX y JSON.

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 más utilizada en la red, es el formato de datos JSON (*JavaScript Object Notation*).

El esquema gráfico siguiente (figura 9) ilustra la arquitectura de las llamadas a servicios mediante este objeto desde JavaScript:

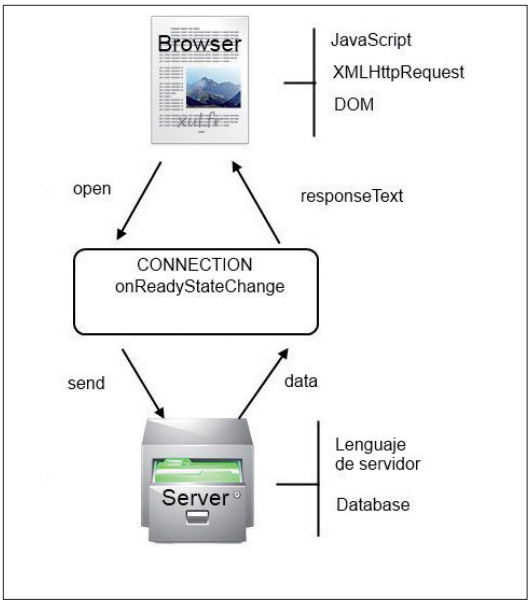


figura 9 Esquema de una llamada *XMLHttpRquest*

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[8].

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, actualizando la interfaz de usuario, pero solo con los nuevos datos o elementos recibidos y no con la página completa.

Previo a la petición, se habrá configurado adecuadamente el objeto **`XMLHttpRequest`**, especificando en su método **`open()`**, el modo en que realizamos ésta (GET, POST,..) y la URL destinataria de la misma. Cuando se recibe la respuesta del servidor, además, algunas de las propiedades 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 de la página siguiente.

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).

[8] http://www.w3.org/TR/XMLHttpRequest2/

```
// 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);
}
```

Como vemos, hemos asignado una función anónima al evento **onreadystatechange**, para que, al ejecutarse, 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:

Forma en que *readyState* devuelve el objeto	**0 = sin inicializar** **1 = abierto** **2 = cabeceras recibidas** **3 = cargando** **4 = completado**

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 del listado de la página siguiente.

Y, en ambos casos, la propiedad **responseText** del objeto **request** almacena la información recibida.

```
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);
}
```

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**[9] 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 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:

[9] Ver la especificación oficial en la página: http://www.json.org/json-es.html

```
<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. Aparte de por seguridad, no es recomendable porque 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 10 al abrir el archivo con cualquiera de los navegadores:



figura 10 Salida del código anterior en IE9/10.

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*.

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 ** ** dentro del cuerpo del documento, de acuerdo con el código de la página siguiente. Véase listado 1.

Para configurar la llamada podemos utilizar un código JavaScript. Véase listado 2.

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 el listado 3.

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.

```
<!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>
```

Listado 1

```
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;
        };
    }
}
```

Listado 2

```
// 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;
    }
}
```

Listado 3

Web Sockets

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, comentaba —en la entrevista cuyo resumen incluimos en el primer capítulo— 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[10].

Y es que, de sus ventajas se derivan también sus inconvenientes: las comunicaciones TCP ordinarias por puertos diferentes al 80 suelen estar bloqueados por temas de seguridad por los administradores, con lo que su uso ofrece una alternativa suministrando una funcionalidad similar a la apertura de varias conexiones en distintos puertos,

[11] En este momento existe una especificación de nivel "Draft" (Borrador) con fecha diciembre de 2011, que puede verse en la dirección http://dev.w3.org/html5/websockets/

pero multi-plexando diferentes servicios WebSocket sobre un único puerto TCP (a costa de una pequeña sobrecarga del protocolo). Pero, por otra parte, a partir de una modificación en el borrador 76 del protocolo[11], 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`).

Existirían, por tanto dos versiones de un nuevo protocolo (`ws://` y `wss://`) según se trate de conexiones estándar o cifradas.

Como consecuencia, no existe aún una especificación final de Web Sockets, si bien se sabe algo de lo fundamental en cuanto a su funcionamiento base, su propósito y cómo debe ser la versión final. Además, respecto a este API, hay dos organizaciones implicadas: la propia W3C y la *Internet Engineering Task Force* (IETF)[12], que se encarga de la normalización del protocolo.

El soporte adicional de Microsoft

Por parte de Microsoft, existe un gran nivel de concienciación sobre la evolución de éste estándar, y el sitio **HTML5Labs** realiza un seguimiento del estándar. No obstante, en diciembre de 2011, dicho sitio anunció oficialmente[13] que no iba a continuar el soporte de dichos prototipos, y recomendaba la implementación de la versión soportada por IE10 hasta el momento.

Además, en el sitio oficial de demos de IE10 de Microsoft[14], disponemos de una demo completa en la que podemos ver su funcionamiento, y probarlo en otros navegadores que lo soportan igualmente, como FireFox, Chrome u Opera.

Paralelamente a estas entidades oficiales, existen otras que se encargan de seguir la evolución del estándar, tales como `www.WebSockets.org` que publica demos de funcionamiento y más información relacionada, entre la que se incluyen ejemplos que podemos usar aquí para ilustrar su funcionamiento, sin complicar innecesariamente el código con la programación de la parte servidora.

---

[11] En su versión 76, para ser exactos.

[12] 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).

[13] En su página dedicada: http://html5labs.interoperabilitybridges.com/prototypes/websockets/websockets/info

[14] http://ie.microsoft.com/testdrive/HTML5/WebSocketsFlipbook/Default.html

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*).

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.WebSockets15.`

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)*.

El API *History*

El API History se define en la especificación oficial[16], e incluye métodos para administrar la pila de historial y la dirección URL de un sitio. Ofrece a los usuarios finales

[15] http://msdn.microsoft.com/en-us/library/hh160729(v=vs.110).aspx

[16] http://www.w3.org/TR/html5/history.html#the-history-interface

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.

Se basa en un objeto del mismo nombre (**history**) que posee algunos métodos útiles para el control de la pila del historial de navegación, como **pushState** y **replaceState**.

Mediante el método **pushState()** se puede crear una nueva entrada en el historial y, opcionalmente, incluir un objeto de estado. Por otra parte, usando **replaceState()**, es posible modificar el elemento activo del historial. La sintaxis de estos dos métodos la ofrece la página oficial del sitio MSDN:

```
history.pushState(data, title, url);
history.replaceState(data, title, url);
```

Donde los 3 parámetros significan respectivamente lo 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.
- **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 es completo, como podemos ver en la siguiente captura del editor:

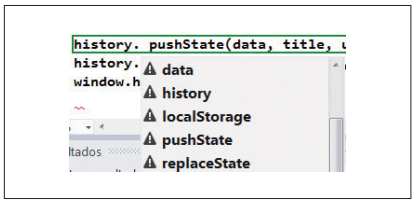


figura 11 Soporte del API History en V. Studio 2012.

Con estos nuevos métodos se puede modificar la ruta o el *hash* (la información que va después de la almohadilla en la dirección URL) pero, por seguridad, no es posible modificar el dominio o el origen de la dirección URL actual.

El evento window.onpopstate

El evento **window.onpopstate** notifica a la aplicación cuando el usuario hace clic en los botones Atrás o Adelante, o cuando se llama al método **history.pushState()** o **history.replaceState()**.

La forma de diferenciar entre los dos en el controlador de eventos es median te una sintaxis como la siguiente:

```
window.onpopstate = function (event) {
    if (event.state) {
        // History ha cambiado mediante
        // pushState o replaceState.
    } else {
        // Carga normal de página
    }
}
```

Naturalmente, el parámetro del evento **popState** contiene el estado del método **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.

Referencias

"*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`.

<title>

HTML5 y CSS3 y JavaScript

</title>

<autor>

Marino Posadas

</autor>

<resumen>

Esta obra aborda el nuevo estándar bajo el supuesto de que el usuario ya conoce los fundamentos del anterior, y desea actualizarse y/o abordar proyectos de desarrollo que impliquen HTML5 y sus tecnologías asociadas: CSS3, JavaScript y las API relacionadas con ellos. El foco es el propio estándar independientemente de la plataforma para la que vaya a ser utilizado.

A lo largo de la obra se utiliza Visual Studio 2012 como herramienta de desarrollo completa que dispone de cobertura total del estándar en sus editores visuales y de código, a nivel de depuración, pruebas y actualizaciones. Se muestra cómo usarlo en esos contextos y aprovechar las nuevas propuestas que la W3C plantea en su iniciativa "Open Web Platform".

</resumen>

</biografia>



Marino Posadas es Consultor Independiente, Redactor de la revista DNM, MVP en C#, MCT, MCPD y MCTS. Esta es su undécima obra sobre tecnologías y la segunda dedicada al estándar HTML. Ha publicado más de 500 artículos en revistas especializadas en papel y "on-line" y ha colaborado como ponente en diversos eventos para programadores en España, el resto de Europa y América, siempre relacionadas con el desarrollo de software.

</biografia>

netalia
just for developers

MVP Microsoft Most Valuable Professional