Este libro revisa desde los fundamentos todo el edificio conceptual sobre el que se basa la construcción de páginas web de cliente: esto es, se centra en aquello que una página web –una vez descargada- es capaz de hacer en el navegador del usuario. Para ello explica en qué se basa el HTML Dinámico, los fundamentos y uso del Modelo de Objetos de Documento y finalmente, analiza el lenguaje Javascript y las capacidades de las Hojas de Estilo en Cascada, como elementos básicos de la construcción dinámica de páginas.

Respecto a los conocimientos previos, es importante que el lector maneje, siquiera básicamente, los fundamentos del lenguaje HTML, y conviene tener conocimiento de algún lenguaje de programación adicional Visual Basic, Delphi, etc.

HTML Dinámico, Modelos de Objetos y Javascript

Marino Posadas y José Luis Hevia

Grupo Eidos

ADVERTENCIA LEGAL

Todos los derechos de esta obra están reservados a Grupo EIDOS Consultoría y Documentación Informática, S.L.

El editor prohíbe cualquier tipo de fijación, reproducción, transformación, distribución, ya sea mediante venta y/o alquiler y/o préstamo y/o cualquier otra forma de cesión de uso, y/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 universo.

El almacenamiento o archivo de esta obra en un ordenador diferente al inicial está expresamente prohibido, así como cualquier otra forma de descarga (downloading), transmisión o puesta a disposición (aún en sistema streaming).

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 universo, de acuerdo con las leyes y convenios internacionales.

Esta obra está destinada exclusivamente para el uso particular del usuario, quedando expresamente prohibido su uso profesional en empresas, centros docentes o cualquier otro, incluyendo a sus empleados de cualquier tipo, colaboradores y/o alumnos.

Si Vd. desea autorización para el uso profesional, puede obtenerla enviando un e-mail fmarin@eidos.es o al fax (34)-91-5017824.

Si piensa o tiene alguna duda sobre la legalidad de la autorización de la obra, o que la misma ha llegado hasta Vd. vulnerando lo anterior, le agradeceremos que nos lo comunique al e-mail fmarin@eidos.es o al fax (34)-91-5017824). Esta comunicación será absolutamente confidencial.

Colabore contra el fraude. Si usted piensa que esta obra le ha sido de utilidad, pero no se han abonado los derechos correspondientes, no podremos hacer más obras como ésta.

© Marino Posadas y José Luis Hevia, 2000
© Grupo EIDOS Consultaría y Documentación Informática, S.L., 2000

ISBN 84-88457-22-7

HTML dinámico, modelos de objetos y JavaScript

Marino Posadas y José Luis Hevia

Responsable editorial
Paco Marín (fmarin@eidos.es)

Coordinación de la edición
Antonio Quirós (aquiros@eidos.es)

Autoedición
Magdalena Marín (mmarin@eidos.es)
Marino Posadas (mposadas@eidos.es)
José Luis Hevia (jhevia@eidos.es)

Grupo EIDOS
C/ Téllez 30 Oficina 2
28007-Madrid (España)
Tel: 91 5013234 Fax: 91 (34) 5017824
www.grupoeidos.com/www.eidos.es
www.LaLibreriaDigital.com

Índice

ÍNDICE **5**

HTML DINÁMICO Y MODELOS DE OBJETOS **9**

CONOCIMIENTOS PREVIOS 9
OBJETIVOS DEL CURSO 9
¿POR QUÉ HTML DINÁMICO? 10
EL WORLD WIDE WEB CONSORTIUM 10
HTML Y LA W3C 10
HTML Dinámico y Modelos de Objetos 11
Lo que aporta el HTML Dinámico 11
La solución: Los Modelos de Objetos 12
El Modelo de objetos de documento (DOM) 13
Programando los objetos de DOM 14
La jerarquía de objetos de DOM 14
Los objetos window y document *15*
EL OBJETO WINDOW 16
Propiedades del objeto window *16*
La propiedad event y el modelo de objetos de HTML Dinámico *16*
Otras formas de asociar código de script a eventos *18*
Ficheros de script independientes *21*
Otras propiedades del objeto event 21
Un manejador genérico de eventos. *21*
EL MODELO DE EVENTOS 22
PROPIEDADES DE *WINDOW* ASOCIADAS AL EXPLORADOR 24
La propiedad history *24*
La propiedad location *24*

La propiedad screen *24*
Temporizadores: los métodos setTimeout(), setInterval(), clearTimeout() y clearInterval() *25*
Métodos de window para creación de ventanas *25*
Cajas de mensaje estándar: *alert()* 26
Cómo solicitar información del usuario: métodos *prompt() y confirm()* 26
Ventanas de Diálogo Modales: showModalDialog() *28*
Apertura de nuevas ventanas del Explorador: método open() 30
LA PROPIEDAD *DOCUMENT* 31
Una vista jerárquica del modelo de objetos DOM *36*
Ejemplo de Creación Dinámica de Tablas mediante el modelo DHTML 38
OTRAS NOVEDADES QUE APORTA DHTML A TRAVÉS DE EJEMPLOS 40
Posicionamiento Dinámico *41*
Filtros y Transiciones *42*
Descarga dinámica de tipos de letra (Font download) *42*
DATOS ENLAZADOS 42
Modelo de funcionamiento de los objetos de enlace de datos. *43*
Ejemplo de uso de un Data Source Object (DSO) *43*
HTML 4.0: CARACTERÍSTICAS E IMPLEMENTACIÓN EN EXPLORER Y NETSCAPE 47
Estándares e implementaciones *47*
SGML y HTML *48*
Comentarios, atributos y caracteres especiales 48
Tipos de datos 48
URLs 49
Colores 49
Multilongitud 49
Tipos de Hiperenlaces 49
Descriptores de Media 49
Nombres de marcos de destino 49
Declaraciones de Tipo de documento 49
La marca <HTML> 50
La marca <HEAD> 50
La marca <TITLE> 50
El atributo <TITLE> 50
La marca <META DATA> 50
La marca <BODY> 51
Elementos identificadores 51
Bloques y elementos “en línea” 51
Marcas de Agrupación 51
Encabezamientos 52
La marca <ADRESS> 52
El Texto: párrafos, líneas y frases 52
El espacio en blanco 52
Frases 52
Citas 52
Párrafos 52
Rupturas de línea 53
Guiones 53
Texto preformateado 53
Cambios en el documento 53
Listas 54
Tablas y Netscape 54
La marca <TABLE> 54
La marca <CAPTION> 54
Columnas, filas y agrupaciones de ambas 54
Hiperenlaces 55

La marca <A> 55
La marca <LINK> 55
Objetos, Imágenes y Applets 55
La marca <IMG> 55
CONCLUSIÓN 55

PROGRAMACIÓN DE HOJAS DE ESTILO EN CASCADA (CSS) 57

INTRODUCCIÓN 57
LAS CSS O CASCADE STYLE SHEETS (HOJAS DE ESTILO EN CASCADA) 58
¿Es todo correcto en las CSS? *58*
Comenzando con las CSS *59*
Sintaxis de las reglas CSS *60*
Bloques 61
Atributos de estilo CSS 61
Estilos CSS 61
Clases y Pseudoclases 62
Tipos de documentos DOCTYPE *63*
Versión "loose" 63
Versión "Strict" 63
Versión "XHTML" 64
Especificación CSS *64*
Curiosidades y nuevos estilos… 80
CASO PRÁCTICO 80
Definición de la práctica *80*
La forma final de la página... ..... *81*
El diseño inicial de la página... ..... *81*
Aplicando estilos. Análisis de requisitos. *83*

INTRODUCCIÓN A JAVASCRIPT 87

INTRODUCCIÓN 87
Oficialidad, Autoría y Estandarización *87*
Javascript y el escenario de ejecución *88*
Objetos de Java Script de servidor *89*
Características del lenguaje Javascript *89*
Sentencias 89
Operadores *90*
De comparación 90
Aritméticos 90
Modulo (%) 90
Operador de Auto-incremento (++) 91
Operador de Decremento (--) 91
Negación (-) 91
Operadores lógicos 91
And (&&) 91
Or (||) 92
Not (!) 92
Operadores de cadenas de texto 92
Operadores de asignación 92
Operadores de desplazamiento 93
And a nivel de bit (&) 93
Or a nivel de bit (|) 93
Not a nivel de bit (~) 94
Operadores especiales 94
Operador condicional (?:) 94
Operador Coma (,) 94

delete 95
new 95
this 95
Tipeof 95
Los Tipos de datos especiales 96
PALABRAS RESERVADAS DE JAVASCRIPT 96
var 97
ESTRUCTURAS DE CONTROL 98
Sentencia if - else *98*
switch 98
Comentarios 99
Bucles while *99*
Bucles do - while *100*
Bucles for *100*
Sentencia break *101*
continue *101*
delete *102*
export *102*
import *103*
function *103*
return *104*
OBJETOS PREDEFINIDOS POR JAVASCRIPT 104
Objeto array *104*
Objeto date *104*
Métodos del objeto Date 105
Ejercicio práctico de date 106
Objeto Math *107*
Propiedades de Math 107
Métodos de Math 107
Ejercicio práctico de math 108
Objeto string *109*
Propiedades de String 109
Métodos de String 109

PRÁCTICAS CON JAVASCRIPT 111

JAVASCRIPT A TRAVÉS DE EJEMPLOS 111
Ejercicios básicos *111*
Construcción y llamada a una función 111
Utilización de cajas de diálogo 112
Las propiedades del objeto window 113
Control de usuarios mediante password 113
Navegabilidad mediante la propiedad *history* 114
Ubicación actual mediante URL, y referencias agrupadas mediante *with* 114
Averiguar la versión y el tipo de navegador utilizado: 115
Creación de un objeto 116
Mas aplicaciones de los objetos: Presentar Fechas y Horas 118
Otro ejemplo, más completo 118
Escritura dinámica de un documento 120
Llamada condicional a una función 121
Ejemplos de uso del modelo de objetos DOM 121
Uso del método *cloneNode* 121
Algunos efectos visuales 124
Degradación de fondos 124
Posicionamiento por capas mediante estilos 124
Algunos sitios web de utilidad para los lectores 126

1

HTML Dinámico y Modelos de Objetos

Conocimientos previos

Esta obra asume que el lector conoce, siquiera superficialmente, el lenguaje básico de construcción de páginas Web (HTML), y preferiblemente (aunque esperamos que no resulte imprescindible), algún otro lenguaje de programación orientado a eventos (Visual C++, Visual Basic, Visual FoxPro, Delphi, etc.

Objetivos del Curso

Este curso pretende revisar los aspectos fundamentales de las tecnologías implicadas en la construcción de páginas web de cliente, esto es, de cómo programar su comportamiento en los navegadores, independientemente de los mecanismos implicados en la generación de dichas páginas en los proveedores de contenidos. O dicho de otro modo, lo que una página puede hacer una vez que es cargada y visualizada por un navegador, y que genéricamente se conoce bajo el nombre común de DHTML o HTML Dinámico.

Esto supone –en el estado actual de las tecnologías- que no sea uno sólo el lenguaje o estándar usado en la construcción de páginas, sino un conjunto de ellos, cada uno de los cuales, con su pequeña historia, su especificación oficial y un comportamiento de cara al desarrollador.

¿Por qué HTML Dinámico?

Lo primero que podemos preguntarnos es qué fue lo que llevo a los grandes distribuidores de software a cambiar el HTML de siempre. La respuesta más inmediata la encontramos en la necesidad de separar los datos del contenido. Hasta ese momento, los dos estaban indisolublemente unidos y no podía disponerse de ficheros de presentación independientes que permitieran cambiar un estilo de presentación simultáneamente en todas las páginas de un sitio web. El otro gran problema era la interactividad. La única interactividad que se permitía se reducía a los hipervínculos. El resto no estaba previsto por un lenguaje que sólo se preocupaba de mostrar contenidos.

Además, el mero hecho de hablar de implementaciones suponía problemas, ya que los fabricantes no estaban de acuerdo en la forma de construir los navegadores. Esto, empezó a resolverse cuando apareció en escena un organismo de normalización que pretendía hacerse cargo de la elaboración de los estándares que se utilizasen en Internet. Ese organismo era la World Wide Web Consortium.

El World Wide Web Consortium

W3C es una entidad de carácter internacional dedicada a la normalización, y creada en 1994, similar en cierto sentido a ISO o ANSI. Sus miembros son más de 400 organizaciones de todo el mundo que corren con los gastos de financiación –aparte de algunos ingresos estatales- y lo componen profesionales de la informática de todos los sectores, cuya misión es la de 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 (www.w3.org) 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 para Internet, distribuidoras y centros de investigación. Baste citar entre sus miembros más destacados a Adobe, AOL, Apple, Cisco, Compaq, IBM, Intel, Lotus, Microsoft, Motorola, Netscape, Novell, Oracle, Sun, y un largo etcétera.

Su Web, la 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., con réplicas en el INRIA (http://www.inria.fr/) en Europa (Francia) y la Universidad de Keio en Japón (http://www.keio.ac.jp/). Hasta el momento, han desarrollado más de 20 especificaciones técnicas, siendo su mentor principal Tim Berners-Lee (inventor de la WWW, y accesible en la dirección http://www.w3.org /People/Berners-Lee) quien hace las funciones de jefe de equipo y trabaja en colaboración con otros equipos del resto de organizaciones miembro.

HTML y la W3C

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 de 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.

HTML Dinámico y Modelos de Objetos

Se dice que **HTML 4.0** es la última y también la mejor de las especificaciones actuales. No obstante, existe una revisión posterior para problemas menores, llamada **HTML 4.01**, pero sólo incluye algunas correcciones a la anterior. Data del 24/Dic/1999, y sobre ella se ha basado una nueva especificación, llamada **XHTML 1.0**, que utiliza sintaxis **XML** para las etiquetas HTML, pero que no es objeto de este libro. Con esta versión final, se pretenden resolver muchos de los problemas que se presentan hoy en día, extendiendo las capacidades del lenguaje en muchas áreas y añadiendo posibilidades más acordes con las necesidades de mercado. Sin embargo, probablemente, será el último, tras una larga agonía. La razón, es que su sucesor, **XML**[1] resuelve muchos de los problemas insolubles por los anteriores, ofreciendo un auténtico estándar para transporte de información, y para presentaciones sofisticadas, con elementos multimedia, tratamiento de datos, etc.

Lo que aporta el HTML Dinámico

Entre las aportaciones más interesantes ligadas a la propuesta DHTML caben destacar las siguientes:

- **Estilos dinámicos**
    - Permiten la definición de características de presentación utilizando un lenguaje de presentaciones estándar definido igualmente por la W3C: Las *Hojas de Estilo en Cascada (CSS)*. Junto con la nueva etiqueta <STYLE>, permite la definición separada de formatos que pueden aplicarse a un número indeterminado de páginas simultáneamente. Permite la redefinición de estilo para etiquetas estándar de HTML o la creación de estilos personalizados de usuario. También se puede tener acceso programático a esos estilos, pudiendo cambiarlos en tiempo de ejecución. Dedicaremos un capítulo completo a su uso y fundamentación más adelante.
- **Contenido Dinámico**
    - Utilizando el modelo de objetos DHTML, es posible cambiar el contenido de un documento después de que ha sido cargado. Esto incluye la inserción y borrado de elementos, la modificación de elementos existentes, y el acceso y modificación de cualquier atributo individual.
- **Posicionamiento y Animación**
    - Entendemos por posicionamiento la capacidad para situar un elemento HTML específico en la ubicación que se desee en la página web, independientemente de la posición del resto de elementos. Esto significa que un elemento puede aparecer situado por delante o por detrás de otro, y que podemos establecer de forma absoluta, mediante coordenadas, su posición en una página, sin dependencia funcional alguna. Esta capacidad es una extensión de las posibilidades de las Hojas de Estilo, y forma parte de un conjunto de posibilidades: desde programación de rutinas de movimiento y establecimiento de intervalos de ejecución, hasta las nuevas etiquetas como <MARQUEE>, o la capacidad de insertar *applets* de Java o componentes ActiveX en las páginas.

[1] El lector puede consultar el e-Book "Introducción a XML" de ésta colección, para información adicional sobre éste estándar.

- **Filtros y Transiciones**
    - Por filtros, entendemos ciertos efectos visuales, como las letras con sombreado, y otros similares. Se consideran también como una extensión de CSS. Las transiciones son, así mismo, efectos, que pueden resultar útiles en ciertas presentaciones de información, como las denominadas *Slide Shows*, o pruebas de muestreo de imágenes encadenadas.

- **Descarga de Tipos de Letra (Font Download)**
    - Consiste en la capacidad de utilizar tipos de letra descargados dinámicamente. Mediante el atributo de estilo **@font-face**, un documento puede hacer referencia a un tipo de letra descargado dinámicamente, y desecharlo inmediatamente después de que el documento sea descartado.

- **Enlace de Datos (Data Binding)**
    - Es la característica que permite enlazar elementos individuales de un documento a orígenes de datos provenientes de bases de datos o de ficheros delimitados por comas. Uno de los usos prácticos de esta posibilidad es la de generar tablas con el contenido de bases de datos de forma automática y dinámica, por ejemplo, enlazando una etiqueta <TABLE> a un origen de datos. Esta función se realiza mediante un objeto DSO (*Data Source Object*), que sabe como conectarse con el origen de datos adecuado. Se implementa en forma de un componente ActiveX o un Applet de Java.

- **Modelo de objetos DHTML**
    - Es la base de todo el HTML Dinámico, y suministra la interfaz que permite a los lenguajes de script y a los componentes acceder a las características de HTML Dinámico. A continuación realizamos una explicación más exhaustiva del modelo de objetos en que se basa.

La solución: Los Modelos de Objetos

Así pues, la propuesta de mejora de HTML se basó sobre todo en dos estándares, las Hojas de Estilo en Cascada, al que dedicamos un capítulo más adelante, pero fundamentalmente, en el **Modelo de Objetos de DHTML**, que posteriormente se estandarizó en el llamado **Modelo de Objetos de Documento (DOM)**, de acuerdo con una especificación oficial de la W3C.

Un Modelo de Objetos es un mecanismo que permite el acceso y la programación de documentos y aplicaciones informáticas. Los modelos de objetos permiten exponer al programador el conjunto de funcionalidades de un programa o de características de un documento en forma de objetos programables.

Por ejemplo, el conjunto de Tablas de un documento de *Microsoft Word*, aparece en el modelo de objetos de documento de Word como una propiedad colectiva (*Collection*) del propio documento (la propiedad *Tables*). Podemos referirnos a cada tabla, a cada celda y cada contenido de celda usando las propiedades de los objetos *Table* de esa colección. Y lo mismo podemos decir para cada palabra del documento, cada estilo usado en la presentación, o cada ítem de menú o Caja de Diálogo disponible en Word. Decimos, por tanto, que Word ha sido construido de acuerdo con un modelo de objetos programable.

Naturalmente, si hablamos de objetos programables, necesitaremos un lenguaje de programación. En el caso citado, ese lenguaje es **VBA** (*Visual Basic para Aplicaciones*), el núcleo de la herramienta Visual Basic que contiene las definiciones básicas del lenguaje y que está disponible como lenguaje de macros, no sólo para toda la "suite" de Office, sino también para otras aplicaciones que han sido proyectadas de ésta forma, como *AutoCad*, *3D-Studio*, *Visio*, etc.

En general una herramienta que soporte VBA como lenguaje de macros está hecha de tal forma que cualquier característica de la herramienta o de los documentos generados con ella, aparece como un elemento programable y accesible a través de VBA.

Vemos, pues, que los modelos de objetos extienden la funcionalidad de las aplicaciones, pues permiten su ejecución indirecta a través de código fuente. Al conjunto de rutinas de carácter público programadas de ésta forma se le llama conjunto de macros de un documento.

El Modelo de objetos de documento (DOM)

Al modelo de objetos diseñado para convertir el lenguaje HTML en DHTML se le denomina DOM (Modelo de Objetos de Documento). Es algo que debe ser aportado por el propio navegador (generalmente, en forma de una DLL que se carga al abrir el programa explorador) y que modifica la forma en que las páginas web son procesadas.

El esquema que muestra la Figura 1 ilustra la forma en la que viaja la información y se transforma de estática en dinámica.

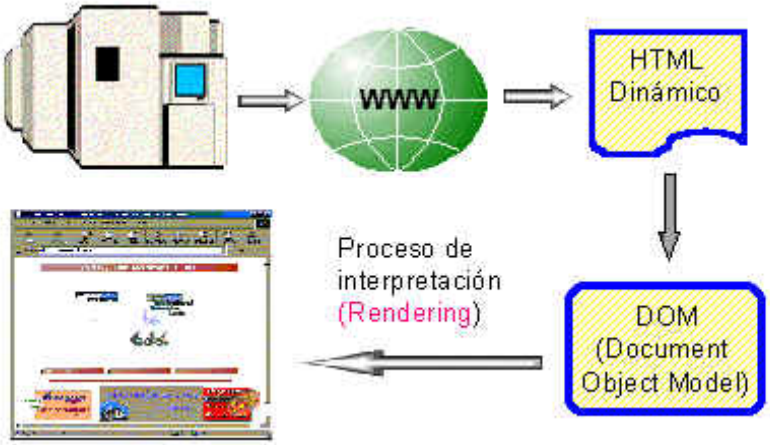

Figura 1. Esquema de envío y transformación de una página web mediante DOM

DOM es lo que convierte el HTML estático en dinámico, y podemos entenderlo como la forma en la que los exploradores interpretan una página desprovista de comportamientos programables, transformando sus elementos en objetos, que, como tales, poseen propiedades, métodos y eventos, y que por lo tanto, se convierten en entidades programables. Esta era la base del dinamismo del nuevo HTML, junto a la incorporación de algunas etiquetas nuevas que permiten aplicar estilos de forma global, como <SPAN> y <DIV>, si bien el tema de los estilos será objeto de un tratamiento independiente en otro apartado.

En el esquema anterior, cuando un usuario solicita una página web (por ejemplo, http://eidos.es) el servidor web busca dicha página, la envía al cliente y allí sufre un proceso de transformación: según se va leyendo el contenido, se construyen tantos objetos en la memoria como etiquetas tenga la página

HTML asignando especialmente las que tengan un identificador (ID ó NAME), y finalmente, se da un formato gráfico de salida al documento, al tiempo que el motor del navegador permanece a la escucha de los eventos que el usuario genere al navegar por la página. Cuando se produce uno, (como pasar el cursor por encima de un ítem de menú, o de un gráfico) el navegador invoca al intérprete del lenguaje de *script* en el que el evento tenga asociada su acción y ésta se ejecuta. Esa es la forma en que los menús cambian de color o de tamaño cuando navegamos por ellos, y también la forma en la que se producen un sinfín de efectos especiales que estamos ya acostumbrados a ver en las páginas web.

Una vez más, es labor del navegador (o de las librerías que realizan la interpretación o *rendering*) el construir objetos dinámicos a partir de lo que sólo es un documento, evitando así, el envío de componentes a través de la web.

Programando los objetos de DOM

Ahora bien, una vez realizada esa labor de conversión, necesitamos un mecanismo de programación, un lenguaje. En el escenario de Internet, a los lenguajes de programación que nos permiten manejar los objetos DHTML, se les denomina **Lenguajes de Script**, y entre los más popularmente aceptados destacan: *Javascript, VBScript, PerlScript y Python*, basados los tres primeros, respectivamente, en sus hermanos mayores, *Java, Visual Basic y Perl*). La forma en que se manipula un objeto DHTML es asignándole un identificador (un valor para su atributo ID ó NAME, lo que lo convierte en objeto programable) y programando una acción escrita en uno de estos lenguajes, que, normalmente, estará asociada con alguno de los eventos de que el objeto disponga.

Es precisamente la definición de ese conjunto de objetos, la que queda establecida mediante DOM, si bien podemos decir que existen dos versiones del modelo. Previamente se definió una normativa basada en colecciones que fue implementada por los navegadores más populares y que se llamó **Modelo de Objetos DHTML.** Todavía está soportada en las versiones más actuales, con lo que en realidad, la inspección de los modelos de objetos nos muestra mecanismos que pertenecen al modelo antiguo y al nuevo conviviendo y pudiendo ser utilizados desde el mismo documento. En principio, tanto **Internet Explorer** como **Netscape**, cumplen con una buena parte de la especificación DOM oficial de W3C, si bien ambos poseen ciertas limitaciones en esa versión, que en el caso de Explorer, han sido mejoradas y completadas en versiones posteriores (la reciente versión *Netscape 6.0*, propugna un soporte completo de DOM, si bien la forma en que soporta la presentación mediante de **Hojas de Estilo en Cascada** se limita a la primera especificación, de nombre **CSS1 -***Cascading Style Sheets Level 1*- y a algunas características de la más moderna **CSS2**). Precisamente por razones de compatibilidad con el estándar, en este curso usaremos genéricamente Internet Explorer para mostrar los resultados de las páginas de ejemplo.

La jerarquía de objetos de DOM

La Fig. 2 muestra el gráfico correspondiente a la jerarquía de objetos de DOM, en su versión inicial, tal y como se soporta por **Internet Explorer 4.0** y posteriores. Podemos apreciar, que como en toda jerarquía dinámica, existe un objeto inicial o raíz del cual penden el resto de objetos: **window**.

Ese objeto *window*, tiene, a su vez, propiedades y métodos que suministran el acceso a las distintas características del navegador y de los documentos que se abren en él. La más importante de esas propiedades es **document**, que hace referencia a la página web abierta en cada momento. Por tanto, podemos resumir esta introducción asumiendo que el objeto **window** controla todo lo relativo al programa navegador, mientras que el objeto **document**, maneja todo lo concerniente a la página activa.

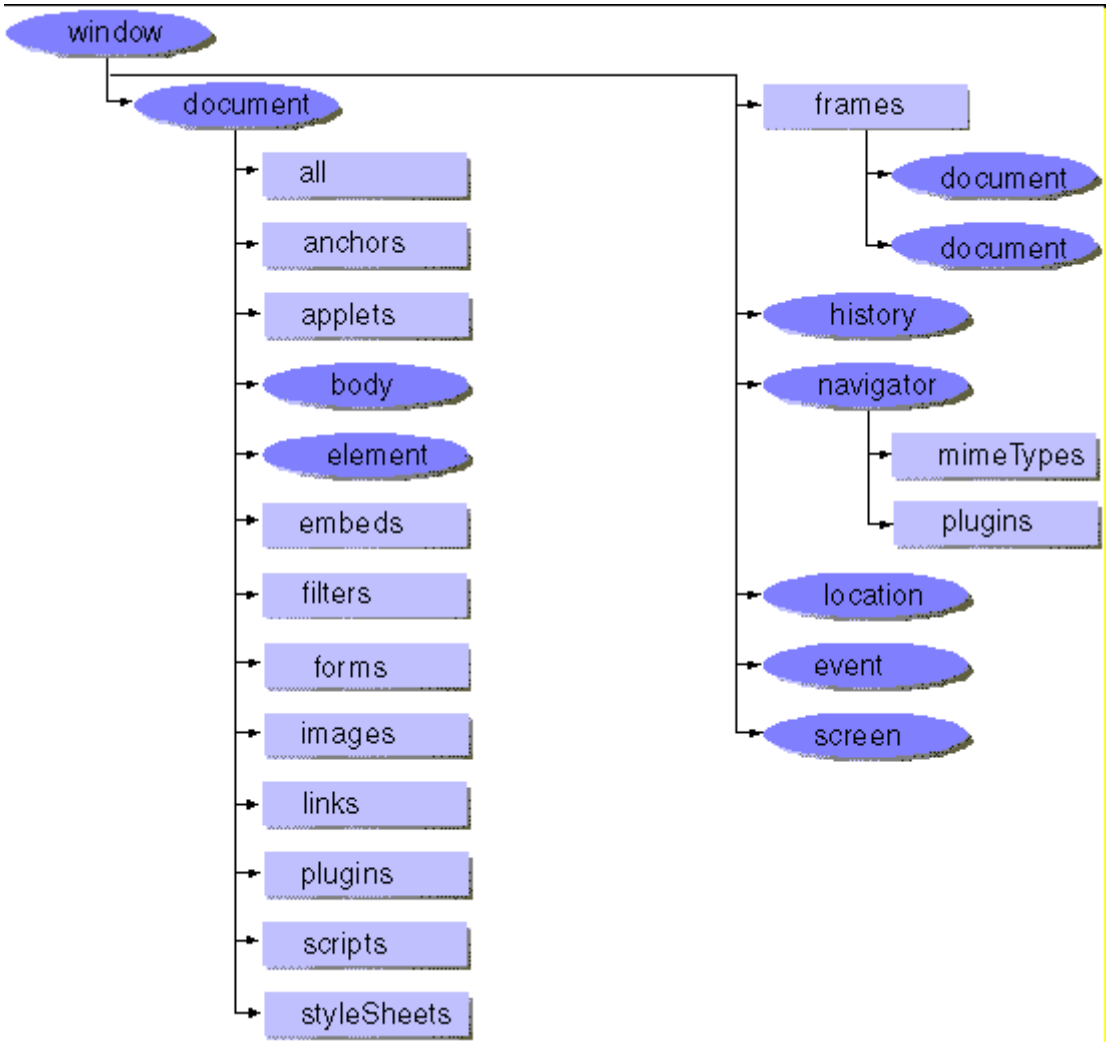

Figura 2. Modelo de objetos DHTML (fuente: MSDN)

Los objetos *window* y *document*

Del esquema se deduce que en la jerarquía, algunos objetos pasan a pertenecer a colecciones concretas, como la colección de imágenes (*images*) o la colección de hojas de estilo (*styleSheets*). Otros, por su importancia se transforman en objetos individuales, como *body* o *element*, y además existe una colección donde van parar todos los objetos del documento: *all* (el objeto *document* es uno sólo, y no una colección, ya que se trata de una interfaz SDI).

Otra característica de la jerarquía de DOM es que en ella encontramos colecciones que deben existir siempre, independientemente de que contengan o no elementos. Por ejemplo, siempre descubrimos una colección **images**, aunque la página no contenga una sola imagen. Podemos hacernos una idea de la forma en que DOM crea la estructura mediante una jerarquía de tipo árbol, donde cada etiqueta se convierte en un nodo. Algunos nodos serán equivalentes a los troncos y su misión principal (aunque no la única) será contener una serie de nodos hijos (*child nodes*), como es el caso de las etiquetas contenedoras del tipo <HTML> ó <BODY>, mientras que otros equivaldrán a las hojas del árbol, sirviendo para albergar los contenidos del documento sin incluir otros nodos hijos (el texto que se encuentra entre las etiquetas del tipo <P>, <SPAN>,<H1>, etc).

Otro matiz a tener en cuenta es que el modelo de objetos **distingue entre objetos programables mediante eventos y aquellos a los que se puede hacer referencia**. **Para que un objeto sea**

programable mediante eventos se requiere un identificador ó ID, mientras que siempre podemos hacer referencia de lectura o asignación de las propiedades de un objeto cualquiera del documento mediante la colección a la que pertenezca.

El objeto window

Ya hemos comentado que todos los objetos de la jerarquía DOM son descendientes del objeto ***window***. Este objeto nos da acceso a todas las características del navegador, pero, a través de algunas de sus propiedades, también tenemos acceso a los elementos del documento. Vamos a revisar algunas de sus propiedades y métodos más interesantes.

Propiedades del objeto *window*

Hemos visto en el esquema que el objeto *window* dispone de 6 propiedades:

1) **event**
2) **frames**
3) **location**
4) **history**
5) **navigator**
6) **screen**
7) **document**

Vamos a revisar algunas de las características de programación más interesantes y utilizadas de estos objetos.

La propiedad *event* y el modelo de objetos de HTML Dinámico

La primera propiedad, *event*, se relaciona directamente con el llamado ***Modelo de eventos de HTML Dinámico***. Se trata -como ya se ha dicho- de un modelo basado en la construcción dinámica de objetos a partir de etiquetas, y que permite la propagación de éstos eventos a sus elementos superiores, (*Sistema de promoción de eventos).* O sea que, la respuesta a un evento, de haberla, puede asociarse al objeto que lo recibe o a cualquiera de sus elementos contenedores, pudiendo llegar –si es necesario- al propio objeto *document.*

Definición: cuando un usuario interactúa con una página o cuando el estado del documento va a cambiar, se activa un evento.

En la práctica, el usuario produce eventos mediante pulsaciones de teclado o de ratón. El sistema puede hacerlo al terminar los procesos de carga de una página, formulario, gráfico y objeto multimedia, pero también mediante un mecanismo de temporización.

El modelo de promoción de eventos tiene dos ventajas principales: por un lado permite el tratamiento global de eventos, asociando una misma acción a cualquiera de los contenidos de un elemento. Por ejemplo, cuando asignamos una acción al elemento <BODY>, cualquier elemento contenido en el cuerpo del documento que reciba el evento, desencadena la acción.

También es posible anular los eventos e incluso anular las acciones predeterminadas que tales eventos puedan provocar. Cabría pensar en situaciones en las que una acción predeterminada como el hecho de seguir un vínculo, puede ser anulado por un objeto contenedor de dicho vínculo. Veamos, por ejemplo, el Código fuente 1.

```
<BODY onclick="return false">
<A name=Enlace_a_EIDOS href="http://www.eidos.es">
<P>Grupo EIDOS</P></A>
</BODY>
```

Código fuente 1

El Código fuente 1, anula el enlace al sitio web de Grupo EIDOS, mediante la asignación de la secuencia '***return false***' al evento *onclick* de la etiqueta <BODY>. El lector puede inferir a partir de aquí las posibles implicaciones dinámicas que esto tiene en la construcción de páginas cuyo comportamiento varíe en función de un nivel de permisos o unas asignaciones de seguridad, por ejemplo: podemos programar situaciones en las que, dependiendo del nivel de acceso, una serie de enlaces permitan o no el acceso a los sitios enlazados.

Ya hemos visto cómo vincular acciones a los elementos mediante la asignación de acciones directas y mediante llamadas a funciones almacenadas en etiquetas <SCRIPT>. Sin embargo, se puede personalizar un <SCRIPT> de forma que sus acciones se asocien exclusivamente a un evento de un objeto. Esto se consigue mediante los atributos FOR y EVENT de la etiqueta. Por ejemplo, podemos asignar una acción al evento **onclick** del botón **B1**, mediante el Código fuente 2.

```
<SCRIPT FOR='B1' EVENT='onclick'>
      Msgbox "Botón pulsado"
</SCRIPT>
```

Código fuente 2

El problema aquí es que las versiones 4.x de Netscape Navigator ignoran estos atributos, intentando ejecutar las acciones del *script* inmediatamente.

Este problema del soporte de navegadores parece que está empezando a diluirse a medida que las compañías distribuidoras de software de navegadores comienzan a basarse en los estándares. Por ejemplo, Netscape 4.x tampoco soporta la propiedad **innerHTML** para los contenidos de los objetos mientras que la versión 6.0 sí lo hace. Lo mismo sucede con los atributos para la identificación de objetos: mientras las versiones 4.x de Netscape solo soportan el atributo NAME, en la versión 6.0 ya se soporta el atributo ID, y en general existe un soporte más próximo al estándar DOM de la W3C.

Se debe de tener cuidado con los nombres de los eventos si no especifica un lenguaje predeterminado o si explícitamente se indica Javascript como lenguaje a utilizar, porque en ambos casos, es el motor de interpretación de Javascript el que se pone en marcha y es sensible a mayúsculas y minúsculas.

Otras formas de asociar código de *script* a eventos

Otra posibilidad para codificar procedimientos dentro de una página web, consiste en utilizar los atributos que DOM asigna a cada elemento cuando lo convierte en un objeto programable.

Una de las transformaciones más importantes que DOM realiza es la de asignar a cada etiqueta dinámica un atributo por cada evento al que es capaz de responder. Para programar una acción, basta con asignar como valor del atributo el nombre de una función escrita en uno de los lenguajes de script válidos, o invocar un método o propiedad de los objetos disponibles en el modelo DOM.

Podemos ver este comportamiento codificando otro efecto muy conocido actualmente, consistente en el cambio dinámico de estilos cuando el cursor pasa por encima de algún elemento.

Se puede utilizar la propiedad ***srcElement***, teniendo en cuenta que se trata de un objeto que hace referencia al elemento que se haya seleccionado, y la característica que hace que cualquier elemento DHTML disponga de una propiedad (***style***), para modificar los valores que determinan su presentación.

Eso significa que todo lo que tenemos que hacer es asignar a los atributos ***onmouseover*** *(pasar el ratón por encima de un área)* y ***onmouseout*** *(abandonar el área)* la ejecución del código correspondiente (nótese que no estamos invocando a ninguna función incluida en etiquetas de *script,* sino que asignamos el código a ejecutar directamente a los atributos del elemento).

Imaginemos que nuestra página dispone de una par de elementos <P> con un cierto contenido y que queremos que cuando el usuario pase el cursor por encima de un de ellos, cambie automáticamente su estilo a color rojo y tamaño de letra de 20 píxeles. También queremos que cuando el cursor salga de la zona correspondiente al elemento, éste vuelva a su estado original.

```
<P style=”font-family:Arial; font-size=14”
onmouseover="window.event.srcElement.style.color='Red'"
onmouseout="window.event.srcElement.style.color = 'Black'">
Primer ítem que cambiará de estilo</P>

<P style=”font-family:Arial; font-size=14”
onmouseover="window.event.srcElement.style.color='Red'"
onmouseout="window.event.srcElement.style.color = 'Black'">
Segundo ítem que cambiará de estilo</P>
```

Código fuente 3

El lector puede copiar el Código fuente 3 fuente a una página cualquiera y comprobar su funcionamiento, como aparece en la Figura 3

En el caso de disponer de una serie de ítem a los que queremos aplicar un comportamiento común, podemos usar ésta técnica junto a la capacidad de promoción de eventos, para programar con un único código ese comportamiento múltiple.

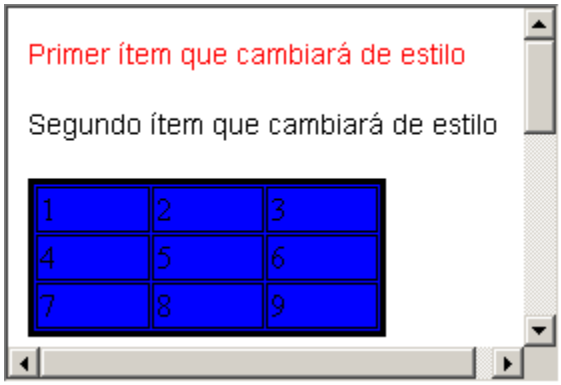

Figura 3. Cambio dinámico de estilo

```
<html>
<head>
<style>
.Normal {
   cursor: hand;
   font-family: verdana;
   font-size: 20;
   font-style: normal;
   background-color: blue;
   color: white;
   display: block
 }
.Resaltado {
   cursor: hand;
   font-family: verdana;
   font-size: 20;
   font-style: italic;
   background-color: white;
   color: blue;
   display: block
 }
</style>
</head>
<body>
<P style="FONT-FAMILY:Arial;font-size:21">Seleccione un Bicho</P><HR>
<span class=Normal>Lagarto Juancho</span><br>
<span class=Normal>Leoncio el León</span><br>
<span class=Normal>Pepe Pótamos</span><br>
<span class=Normal>Gorila Magilla</span><br>
<span class=Normal>Gato Silvestre</span><br>

<script>
// Asignamos comportamiento por defecto al objeto document
//lo que se aplica a cada uno de sus nodos hijos.
document.onmouseover = MostrarEspecial;
document.onmouseout = MostrarNormal;

function MostrarEspecial() {
  //si el estilo es Normal, cambiar a Resaltado
  if (window.event.srcElement.className == "Normal") {
     window.event.srcElement.className = "Resaltado";
  }
}

function MostrarNormal() {
  //si el estilo es Resaltado, cambiar a Normal
  if (window.event.srcElement.className == "Resaltado") {
     window.event.srcElement.className = "Normal";
```

```
  }
}
</script>
</body>
</html>
```

Código fuente 4

Aunque estudiaremos las Hojas de Estilo más adelante, anticipemos para este ejemplo que, lo mismo que hemos definido una presentación asignando a un atributo ***style*** unos valores, podemos separar esas definiciones de estilo, situándolas esta vez dentro de una etiqueta <STYLE>, permitiendo que sean accesibles a todo el documento. Cuando operamos así, la manera de aplicar uno de los estilos definidos es mediante el atributo ***class*** del elemento, tal y como aparece en las definiciones iniciales.

```
<span class=Normal>Lagarto Juancho</span>
```

Código fuente 5

En el caso de que la asignación de estilo sea dinámica (en tiempo de ejecución), la propiedad ***className*** de cada elemento será la que nos permita realizar la nueva asignación.

En el ejemplo, utilizamos la definición de dos estilos de usuario (**Normal** y **Resaltado**). Observe el lector la asignación inicial al objeto *document* de las acciones a realizar cuando el cursor pasa por encima de un elemento (*onmouseover*) y cuando sale de él (*onmouseout)*. Utilizamos la propiedad *className* del objeto document, que es el equivalente del atributo *class* para etiquetas de HTML Dinámico. Como cada elemento genera sus propios eventos, si no tienen ninguna acción asociada, se produce la promoción del evento hasta llegar a algún elemento de orden superior que **sí** tenga una acción asociada: **document**, en este caso. Entonces, se determina cual es el elemento que ha producido el evento y se le trata aplicándole el estilo correspondiente.

Figura 4. Cambio dinámico de estilos aplicado de forma global.

También podemos incluir elementos ocultos, que solo se muestren tras la ejecución de un evento. El estilo de presentación ***display***, puede adoptar los valores "*hidden"* (oculto, los elementos no se visualizan) o vacío "", que significa que se permite la visualización. Veremos muchas de estas características en el capítulo dedicado a las hojas de estilo en cascada.

Ficheros de *script* independientes

Con idea de crear rutinas genéricas que puedan ser utilizadas en diversos documentos sin necesidad de duplicar el código también es posible situar las rutinas en ficheros independientes. La gran ventaja es que al residir en un fichero aparte, conseguimos crear auténticas librerías de *scripts* que pueden ser utilizadas dentro de un proyecto para diferentes páginas web. Suele asignársele la extensión asociada con el lenguaje en que están escritas, generalmente **.JS** (para Javascript) y **.VBS** (para VBScript). Lo único que se precisa es hacer referencia a la librería mediante el establecimiento de un vínculo, usando el atributo SRC de la etiqueta <SCRIPT>.

```
<SCRIPT LANGUAGE="Javascript" SRC="Librería.js"></SCRIPT>
```

Código fuente 6

Otras propiedades del objeto event

Además de la propiedad *srcElement*, ***event*** posee propiedades adicionales para informar al programador de características adicionales del evento que se produce. Por ejemplo, la propiedad *cancelBuble*, permite anular la promoción del evento hacia los objetos superiores en la jerarquía. Por defecto, este valor es *false*. En caso de asignarle un valor *True*, se detiene dicha promoción, pero sólo para la instancia actual de evento, no para otros eventos que puedan producirse.

Otra propiedad interesante es *returnValue*, que sirve para anular la acción predeterminada de un evento, cuando asignamos el valor *false* a ésta propiedad. Observe el lector que el mecanismo de anulación del evento utilizado en el ejemplo anterior, se basa en una palabra reservada de Javascript (*return)*, que actualiza la propiedad *returnValue cuando el controlador del evento devuelve el control al Explorador.*

Un manejador genérico de eventos.

Otras propiedades interesantes del objeto *event*, tienen que ver directamente con las características de producción del evento: por ejemplo, *type***,** devuelve el nombre del evento sin el prefijo ***on***, lo que permite crear manejadores genéricos de eventos (rutinas que determinen el tipo de evento que se ha producido y dependiendo del que se trate, programen diferentes acciones).

Pongamos que se desea determinar el tipo de respuesta asociada a un mismo objeto, pero subordinada al evento producido: por ejemplo, queremos que una caja de texto muestre dos respuestas distintas dependiendo de si pulsamos el botón izquierdo o derecho del ratón: un manejador de eventos que distinga esa situación basándose en la propiedad *button* del objeto *event*, adoptaría la forma que muestra el Código fuente 7.

```
function ManejadorEventos() {
      //Primero miramos si se trata de un evento de ratón usando la propiedad
```

```
        //button
        switch (event.type) {
                case 'click':
                        alert('Hola desde onclick');
                        break;
                case 'mousedown':
                        if (event.button == 1) alert('Hola desde mousedown')
                        else alert('Has pulsado el botón derecho');
                        break;
        }
}
//y en otra parte del código...
<BODY onclick= "ManejadorEventos()" onmousedown= "ManejadorEventos()">
```

Código fuente 7

Hay que significar que el evento *onmousedown*, tiene precedencia sobre el evento *onclick*, por lo que éste último no se ejecuta dentro de esta rutina tal y como se presenta aquí. Si anulamos, la primera opción de comprobación del evento *onmousedown* (*if (event.button == 1) alert('Hola desde mousedown')*, comprobaremos como entonces **sí** que se ejecuta el código del evento *onclick* al pulsar el botón izquierdo y el del evento *onmousedown*, al pulsar el botón derecho.

Naturalmente, necesitamos conocer qué significan los valores devueltos por el evento para poder codificar un comportamiento adecuado. De acuerdo con el manual de referencia, las propiedades asociadas a valores del ratón o de teclado para el objeto ***event*** son los que se indican en la Tabla 1.

Propiedades de event para valores del ratón y teclado		
Button	0	Botón NO pulsado
	1	Botón izquierdo pulsado
	2	Botón derecho pulsado
ctrlKey	True/False	Estado de la tecla CTL
altKey	True/False	Estado de la tecla ALT
shiftKey	True/False	Estado de la tecla Mayúsculas.

Tabla 1. Propiedades para el control de estado de las teclas de ratón, CTL, ALT y MAY.

Con esto presente, pueden programarse acciones complejas o comportamientos especiales que requieran combinaciones de teclas, como por ejemplo, información de seguridad, acceso a socios, etc.

El modelo de eventos

Todos los elementos de un documento, comparten una serie de eventos comunes. Además, aquellos elementos que pueden recibir el foco disponen de una serie de eventos adicionales. La Tabla 2 lista los eventos comunes a todos los elementos de un documento:

Modelo básico de eventos			
Teclado	Teclado	Ratón	Ratón
onkeydown	ondblclick	onmousedown	onmouseover
onkeypress	onclick	onmousemove	onmouseout
onkeyup		onmouseup	

Tabla 2. Modelo básico de eventos

Aquellos objetos capaces de recibir el foco pueden generar los eventos: **onenter**, **onexit**, **onfocus**, y **onblur.** Los dos primeros se producen cuando el ratón entra en el área de documento definida por un elemento, mientras que los dos siguientes les pueden producir el ratón o el teclado al recibir el foco un elemento, para similares circunstancias.

El lector podrá encontrar información específica para cada uno de los eventos disponibles en el modelo, tanto en el MSDN de Microsoft, como en la documentación de cualquiera de los productos de desarrollo para Internet. No obstante, a continuación incluimos una tabla con el modelo completo de eventos:

Modelo completo de eventos					
onabort	onactivate	onafterprint	onafterupdate	onbeforecopy	onbeforecut
onbeforedeactivate	onbeforeeditfocus	onbeforepaste	onbeforeprint	onbeforeunload	onbeforeupdate
onblur	onbounce	oncellchange	onchange	onclick	oncontextmenu
oncontrolselect	oncopy	oncut	ondataavailable	ondatasetchanged	ondatasetcomplete
ondblclick	ondeactivate	ondrag	ondragend	ondragenter	Ondragleave
ondragover	ondragstart	ondrop	onerror	onerrorupdate	onfilterchange
onfinish	onfocus	onhelp	onkeydown	onkeypress	Onkeyup
onlayoutcomplete	onload	onlosecapture	onmousedown	onmouseenter	onmouseleave
onmousemove	onmouseout	onmouseover	onmouseup	onpaste	onpropertychange
onreadystatechange	onreset	onresize	onresizeend	onresizestart	Onrowenter
onrowexit	onrowsdelete	onrowsinserted	onscroll	onselect	onselectionchange
onselectstart	onstart	onstop	onsubmit	onunload	

Tabla 3: Modelo completo de eventos soportado por Internet Explorer 4.0 y posteriores.

Algunos de estos eventos son específicos del navegador Internet Explorer, aunque la mayor parte se soportan por los dos navegadores.

Para concluir con éste apartado dedicado al objeto *event*, conviene recordar que algunos valores dinámicos relacionados con el evento producido, como la posición del ratón en pantalla, también son expuestos al programador en forma de propiedades del objeto *event.* Podemos conocer de esa forma las coordenadas de posición del ratón según se mueve por pantalla en valores absolutos (***screenX, screenY)***, o relativos al navegador (***clientX, clientY)*** o al objeto seleccionado (***x,y***), o incluso saber desde que elemento se desplaza el cursor cuando pierde el foco (***fromElement***) o hacia qué elemento se desplaza (***toElement).***

Propiedades de *window* asociadas al Explorador

En el conjunto de propiedades del objeto *window*, algunas se refieren directamente a las características de la ventana que muestra el navegador. Por ejemplo, los mensajes de la barra de estado se pueden manejar mediante las propiedades *defaultStatus* y *status.* La primera determina el mensaje inicial (por defecto) del navegador, mientras que la segunda se permite la asignación de mensajes temporales.

La propiedad history

Mediante la propiedad *history* podemos navegar hacia delante y hacia atrás dentro del conjunto de páginas visitadas. Se trata de un objeto que dispone de los métodos *back* y *forward*, para navegar hacia la página anterior y la siguiente, respectivamente. Este objeto también dispone de una propiedad *length* que indica el número de páginas visitadas y permite un control de errores.

La propiedad location

Se trata de un objeto que almacena información acerca de la URL que se está visitando. Dado que el formato genérico de una URL es *protocolo://equipo:puerto/ruta?búsqueda#hash*, *location* trocea ese contenido en propiedades más sencillas de utilizar. Des esta forma, la propiedad *host* devuelve el **equipo**, seguido por dos puntos y el **puerto.** La propiedad **href** contiene el URL completo como una cadena única.

Location, dispone de métodos para el manejo de URLs. Por ejemplo, el método ***reload([force])***, permite volver a cargar una página, pudiendo forzar su carga incluso si el servidor informa de que la página no ha cambiado, y el método ***replace(URL)*** carga una página nueva. Funciona como la propiedad ***href*** salvo que la página no se añade a la lista del historial.

La propiedad screen

Proporciona información acerca de la pantalla del usuario, lo que permite al código adaptarse a esas circunstancias si fuera conveniente. Posee las propiedades ***width*** (anchura), ***height*** (altura), ***availWidth*** (anchura disponible) y ***availHeight*** (altura disponible), además de ***colorDepth*** (Bits por punto de pantalla).

Temporizadores: los métodos setTimeout(), setInterval(), clearTimeout() y clearInterval()

Se trata de un método que permite la asignación de un temporizador a la ventana de navegación. Un temporizador, dispara un evento una vez que ha transcurrido un tiempo determinado por código.

Por ejemplo, se puede implementar una rutina para que una vez transcurridos 5 segundos la página activa se traslade automáticamente a otra mediante una rutina similar a la que se muestra en el Código fuente 8.

```
<SCRIPT LANGUAGE="Javascript">
       var Segundos = 5;

       function DejarPagina() {
              if (Segundos == 0) //Cambiamos de página
                  document.location = 'Nueva página.htm';
              else
                //Aquí se hace la cuenta atrás y se visualiza el tiempo transcurrido
                //en la etiqueta <P Id='CuentaAtras'>
                Segundos -= 1;
                document.all.CuentaAtras.innerHTML = 'Quedan ' + Segundos + '...'
                setTimeout('DejarPagina', 1000);
       }
}
</SCRIPT>
```

Código fuente 8

El Código fuente 8, utiliza el método *setTimeout* para establecer un temporizador que se ejecutará cada 1000 milisegundos, lanzando la ejecución de la función DejarPagina() y en cada ejecución resta una unidad a la variable Segundos. Cuando ésta es 0, se navega otra ubicación mediante la propiedad ***location*** del objeto ***document***. Aunque el objeto ***CuentaAtras*** es una etiqueta <P>, podría ser cualquier otro elemento que poseyera una propiedad innerHTML para ir mostrando el tiempo transcurrido.

Métodos de *window* para creación de ventanas

Hasta ahora, hemos utilizado casi siempre el lenguaje VBScript para mostrar cajas de mensaje con el contenido de los elementos que seleccionábamos. La razón es, por un lado, mostrar que los navegadores pueden ejecutar distintos lenguajes de script, y por otro, que no hemos llegado todavía al capítulo dedicado a Javascript. Aunque este procedimiento sea válido a efectos didácticos, la metodología estándar para hacerlo no es esa, por dos razones: primera, por que se prefiere utilizar el lenguaje estándar Javascript de cara a las páginas de cliente, ya que es compatible con casi todos los navegadores (Netscape no soporta VBScript). En segundo lugar, por que es preferible utilizar siempre que sea posible el modelo de objetos en lugar de recursos que pertenezcan a un lenguaje dado.

El objeto *window* dispone de varios métodos asociados con la producción de ventanas, tanto de sistema, como personalizadas. En el primer caso, para la presentación de cajas de mensaje, el objeto *window* dispone de varias posibilidades*:* los métodos ***alert(), confirm()*** y ***prompt()***. Mientras ***alert()*** presenta una caja estándar de sistema, a la que se le puede pasar como argumento la cadena a mostrar (es equivalente al **Msgbox** de VBScript), ***confirm()*** presenta una caja similar, pero con dos botones de confirmación (*Aceptar* y *Cancelar*), pudiendo recoger la selección del usuario en una variable.

Finalmente, ***prompt()*** pide una entrada de datos al usuario y devuelve el dato introducido como valor de retorno (equivale a **InputBox** en VBScript).

Para el caso de que queramos mostrar cajas no estándar (Ventanas de Usuario), disponemos de los métodos ***createPopup(), showHelp(), showModalDialog()*** y ***ShowModelessDialog()***, que cumplen esa función. Incluso podemos abrir otra ventana del propio explorador con una página adicional, como a menudo se utiliza actualmente para presentar pantallas de tipo publicitario u ofrecer contenidos añadidos, usando el método ***open()***. Vamos a revisar brevemente estas posibilidades.

Cajas de mensaje estándar: *alert()*

El objeto *window* dispone además de una propiedad especial, ***event***, que se actualiza automáticamente con información relativa al evento que acaba de tener lugar en el proceso de visualización de una página. Por ejemplo, podemos acceder a la misma información que leíamos antes a través del objeto *document*, utilizando una propiedad del objeto *event.* En concreto, *event* dispone de una propiedad llamada ***srcElement***, que almacena los datos pertinentes al objeto que es receptor del evento: una tabla, una celda, un gráfico, etc. Veamos algún ejemplo de su uso.

Podemos sustituir el código de acceso a los elementos mediante el Código fuente 9, con el Código fuente 10.

```
Msgbox document.documentElement.outerHTML
```

Código fuente 9

```
alert(window.event.srcElement.innerHTML)
```

Código fuente 10

y obtendríamos resultados casi idénticos. Con casi idénticos, queremos decir que –en algunos casos- la precisión es mayor con éste método, pues el ejemplo anterior reconocía las tablas, las celdas y algunos objetos más, pero englobaba al resto de objetos como elementos de <BODY>. Con este sistema se reconoce **cada** elemento individual, obteniendo un control más fino de los elementos de la página.

Cómo solicitar información del usuario: métodos *prompt()* y *confirm()*

El método ***confirm()***, se diferencia de ***alert()*** en que presenta al usuario una alternativa de selección, que podemos recoger en una variable de tipo *boolean*. Por ejemplo, al pedir confirmación para una acción escribiríamos lo que se muestra en el Código fuente 11. Y se presentaría la caja de mensaje que muestra laFigura 5

```
<SCRIPT Language="Vbscript">
       Sub PedirConfirmacion()
       Dim Respuesta
              Respuesta = window.confirma('¿Acepta las condiciones?')
              If Respuesta then
                 'Rutina para el tratamiento de la Respuesta
              End if
```

```
      End Sub
</SCRIPT>
```

Código fuente 11

Por su parte, el método *Prompt()*, equivale a un *InputBox()* en VBScript. Podemos combinar esta capacidad de solicitud de datos junto con el recurso de los contenidos dinámicos para crear páginas de presentación y/o de respuesta, dependiendo de los datos recibidos. Una forma sencilla de codificar esto sería plantear una página que en la entrada (evento *onLoad()*, de la página) pidiese al usuario su nombre, para mostrarlo a continuación, gracias a la propiedad *innerHTML* de cualquier elemento. El Código fuente 12 conseguiría ese resultado.

Figura 5. Caja de Diálogo del método *confirm()*

```
<SCRIPT Language="Vbscript">
      Sub PedirDatos()
            Dim Respuesta
            Respuesta = window.prompt('Introduzca su nombre','Usuario')
            PNOMBRE.innerHTML = Respuesta
      End Sub
</SCRIPT>
<BODY onLoad="PedirDatos">
<!--  y en cualquier otro elemento <P>...
<P ID="PNOMBRE">Aquí va el nombre del usuario</P>
</BODY>
```

Código fuente 12

En tiempo de ejecución, al cargarse la página, el usuario recibe la caja de mensaje que muestra la Figura 6.

Figura 6. Ventana de solicitud de datos al usuario

Y al aceptar la entrada, el valor introducido se almacena en la variable Respuesta, que podrá asignarse a la propiedad innerHTML de cualquier elemento para mostrar la información. Dejamos como

ejercicio de aplicación para el lector implementar ésta solución mediante un botón y una caja de texto que formen parte de un formulario HTML.

Ventanas de Diálogo Modales: *showModalDialog()*

Dentro de las ventanas de usuario, quizá el método más interesante es *showModalDialog()*, ya que nos permite mostrar en otra ventana el contenido de una página web creada por nosotros. La diferencia con su homónima *showModelessDialog()*, es que, mientras la primera obliga al usuario a que la ventana presentada se cierre para continuar con la navegación, no sucede así con la segunda. En cuanto a sus características, la ventana utilizada no es redimensionable y carece de menús y de barras de botones o barras de estado, adoptando el aspecto que puede verse en la Figura 7.

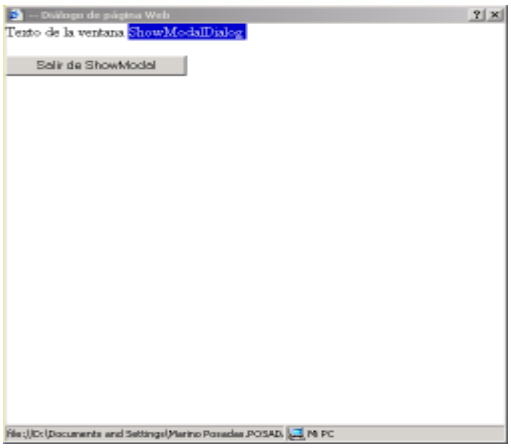

Figura 7. Ventana de tamaño fijo mostrada por el método *ShowModalDialog()*

Respecto a los otros dos tipos de ventanas comentados, *createPopup()* sólo está disponible para las versiones Internet Explorer 5.5 y posteriores.

No obstante, estas ventanas tienen dos inconvenientes principales:

- Son de sólo lectura: El usuario no puede seleccionar los contenidos que se visualizan en la ventana

- No son navegables: Cuando se pulsa en un vínculo no se abre la página en dicha ventana, sino que se ejecuta una nueva instancia del navegador.

En cuanto al paso de parámetros de y desde una ventana de explorador a un cuadro de diálogo, el segundo argumento de la llamada a *showModaldialog()*, admite una variable (que eventualmente puede ser un array) y que se recoge a través de la propiedad *dialogArguments* de la nueva ventana. Recíprocamente, la propiedad *returnValue* de la nueva ventana permite devolver información la ventana principal que efectúa la llamada.

Vamos a implementar un ejemplo de funcionamiento mediante una página que solicite datos personales a un usuario. La ventana principal puede incluir el Código fuente 13, que tendría el aspecto aproximado de la Figura 8.

```
<SCRIPT LANGUAGE="VBScript">
Sub LanzarVentana()
```

```
    Path = "C:/Documents and Settings/mposadas/Escritorio"
    ValorDevuelto = window.showModalDialog Path & "/Formulario.htm", Cstr(Time)
    TxtRespuesta.value = ValorDevuelto
End sub
</SCRIPT>
<INPUT id=button1 name=button1 type=button value="Formulario de Entrada"
 onclick="LanzarVentana()">
<INPUT id=txtRespuesta name=txtRespuesta></P>
```

Código fuente 13

Figura 8. Página que lanza un formulario de entrada mediante *showModalDialog*() y le envía como argumento la hora del sistema.

Y dentro del formulario, basta con incluir un mecanismo de lectura, que situamos en el proceso de carga del formulario, y que muestra la hora en que se lanza y otro de reenvío de la información, en el momento de salir de la Caja de Diálogo y volver a la página que recoge el valor de la primera caja de texto (la que guardaría el nombre del usuario) hace la llamada:

```
<BODY bgColor=silver onload = "Hora.innerHTML = Hora.innerHTML + ': ' +
window.dialogArguments">
---------------------------------------------------------------------------
<INPUT id=submit1 name=submit1 type=submit value=Enviar onclick="Salir()">
---------------------------------------------------------------------------
<SCRIPT LANGUAGE="Javascript">
       function Salir() {
           window.returnValue = window.document.all.text1.value;
          window.close();
       }
</SCRIPT>
```

Código fuente 14

El aspecto del formulario es muy simple, y –dado su carácter modal- no permite que tome el foco la ventana del Explorador hasta que no se haya cerrado (ver Figura 9).

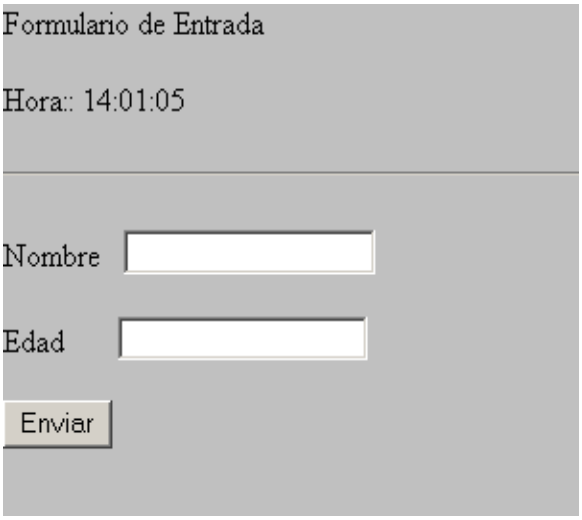

Figura 9. Formulario Modal lanzado con *showModalDialog()*

Añadimos, además, otra página HTML, para que recoja los datos que le envía el formulario principal, y se muestre al pulsar el botón *button1*. En este caso, el dato pasado es la hora en que se entra en la página, y el método de paso, su inclusión como segundo argumento de la función *showModalDialog(), Cstr(Time)*.

La página del formulario, por su parte, recogería esa información leyéndola de su propiedad *dialogArguments* y mostrándola en un elemento:

Al seleccionar el Formulario de Entrada, se mostraría la nueva caja de diálogo modal, cuyo código fuente incluye un mecanismo de lectura para el parámetro pasado, y otro para devolver el valor de retorno incluido en la caja de texto.

```
<BODY bgColor=silver onload="Hora.innerHTML=Hora.innerHTML
+ ': ' +window.dialogArguments">
<P>Formulario de Entrada</P>
<P ID="Hora">Hora:</P>
```

Código fuente 15

Apertura de nuevas ventanas del Explorador: método open()

Respecto al método *open()* nos permite abrir nuevas ventanas del explorador con información adicional y control de la presentación y características de la ventana (control de presencia de bordes, barras de herramientas y estado, menús, etc.) pudiendo, eventualmente ser cerradas desde la ventana que las llamó, gracias a que en el momento de la creación, es posible asignar una referencia a la ventana en una variable.

Posteriormente, es posible llamar a los métodos de esa ventana, como si fueran los de la ventana de ejecución principal (obviamente, habrá que tener cuidado y gestionar los errores que podrían producirse al cerrar el usuario manualmente la ventana e intentar nosotros posteriormente hacer uso de ese objeto inexistente).

Como ejemplo de utilización, podemos abrir una nueva ventana que muestre la misma página web de prueba que hemos utilizado antes en la caja de diálogo, sólo que esta vez se abre en otra instancia del explorador (Código fuente 16), que produce la ventana de salida que muestra la Figura 10.

```
var Ventana2 = window.open('pagina.htm', 'Ventana2',
'width=300,height=250,status=yes,menubar=yes,scrollbars=no,toolbar=no')
```

Código fuente 16

Figura 10. Ventana de explorador mostrada por el método *open()*

Observamos como las características indicadas en el tercer argumento, configuran la forma en que se muestra la ventana (con menús y barra de estado, pero sin barras de desplazamiento, ni de herramientas), y que podemos cerrar a continuación invocando al método *close()* de la ventana a través de la variable 'Ventana2': dentro de la misma ventana y no fuera.

```
Ventana2.close()
```

Código fuente 17

La propiedad *document*

Veamos un ejemplo de lo que estamos comentando. Lo que sigue es una página web muy simple, que posee dos tablas, de las cuales, solamente la primera posee un atributo **ID** de valor "Tabla", que incluye también dos elementos <DIV>, y un gráfico, tal y como aparece en la Figura 11.

1	2	3
4	5	6
7	8	9

Este es un elemento <DIV> que contiene una imagen:

11	12	13
14	15	16
17	18	19

Este es otro elemento <DIV>

Figura 11. Página web simple con 2 tablas y dos etiquetas <DIV>, una de ellas incluyendo un gráfico.

Nota: Aunque existen muchos y buenos editores HTML, para seguir el razonamiento que estamos usando en la construcción de este libro, nos hemos guiado por el entorno de desarrollo de **Microsoft Visual Interdev**, por que muestra los objetos según se crean y permite la aplicación de la tecnología *IntelliSense* al estilo de otras herramientas de desarrollo, mostrando para cada objeto creado la lista de propiedades y métodos disponibles.

Según lo indicado, el acceso desde código a las etiquetas que componen esta página puede hacerse mediante las colecciones individuales a que pertenecen o bien mediante la colección genérica **all**. Si incluimos algo de código fuente asociado al evento *onClick* de la página, podemos acceder y mostrar cualquier parte de su contenido, ya que todo él está expuesto como parte del Modelo de Objetos de DOM. Además, y como se ha comentado, no es imprescindible utilizar el lenguaje Javascript (siempre que asumamos que estamos visualizando el contenido con **Internet Explorer**).

Por ejemplo, dado que el objeto *document* hace referencia a todo el documento, podemos usar la propiedad ***activeElement*** para determinar el objeto en que está situado el cursor (o bien sobre el que se ha producido una pulsación de ratón por parte del usuario), y mostrar el contenido de ese elemento en una caja de diálogo, utilizando la propiedad ***innerHTML***, que, lógicamente, variará en función del elemento seleccionado. El Código fuente 18, muestra diferentes contenidos según pinchamos en distintas áreas de la página.

```
<SCRIPT LANGUAGE="VBScript">
      Sub Sacar()
            Msgbox document.activeElement.innerHTML
      End Sub
</SCRIPT>
<BODY onclick="Sacar()">
```

Código fuente 18

Si hacemos clic sobre el borde de la tabla, estaremos seleccionando uno de los dos objetos *TABLE* de la página por lo que tendremos una caja de diálogo con todo el contenido de la misma (se excluye la propia etiqueta <TABLE>).

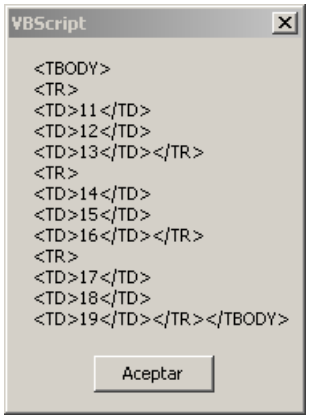

Figura 12. Caja de Diálogo mostrando el contenido de la etiqueta <TABLE>

Pero si seleccionamos solamente una celda, veremos el contenido de la celda:

Figura 13. Caja de diálogo mostrando el contenido de una etiqueta <TD>

Figura 14. Caja de Dialogo mostrando el contenido completo de la etiqueta <BODY>

Y si pulsamos en cualquier otra parte del documento, obtendremos el contenido de su elemento contenedor, esto es, el contenido de <BODY> en este caso (Figura 14)

La propiedad **innerHTML**, muestra el contenido de cualquier elemento y tiene una homónima, **outerHTML**, que hace lo mismo, pero **incluyendo** las propias etiquetas del elemento. Eso quiere decir que si cambiamos según esto la propiedad en el código fuente, (document.activeElement.outer-HTML), al seleccionar la misma celda anterior la salida sería la que muestra la Figura 15.

Figura 15. Nueva salida mostrando la propiedad **outerHTML**

Además de éstas dos, existe una propiedad **innerText**, que muestra los contenidos de cualquier etiqueta, pero *excluyendo todas las etiquetas*, y no sólo las que definen el elemento seleccionado. Esto significa que un nuevo cambio a nuestro código fuente (document.activeElement.innerText), produciría las siguientes salidas:

Para una tabla (Figura 16)

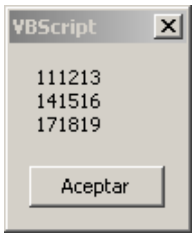

Figura 16. Contenido de una tabla según la propiedad **innerText**

Para una celda coincidiría con los anteriores, mientras que para la etiqueta <BODY>, tendríamos lo que muestra la Figura 17.

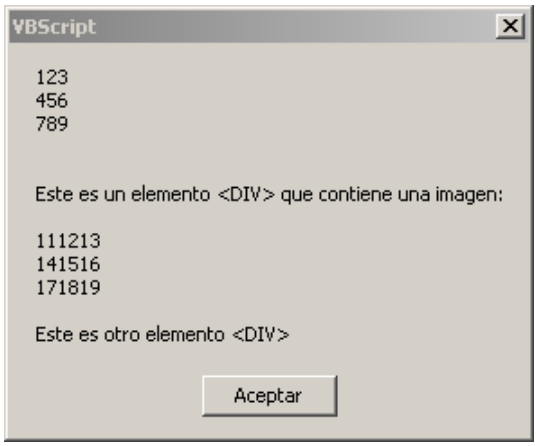

Figura 17. Salida del contenido de <BODY> mostrado por la propiedad **innerText**

Por tanto, vemos que todo depende del objeto **document**, y que la jerarquía está estructurada dependiendo de él. Concretamente, y ya que todo el documento HTML tiene una estructura de árbol, siempre existirá un nodo raíz o principal, que en éste caso, será la propia etiqueta HTML. Ese elemento principal, aparece en la referencia del modelo de objetos como una propiedad del objeto **document**, llamada **documentElement** (el lector tendrá que ser cuidadoso a la hora de nombrar los elementos dentro de su código fuente, pues dependiendo del lenguaje utilizado tales objetos pueden no ser reconocidos en los lenguajes que diferencian entre mayúsculas y minúsculas, como es el caso de Javascript). A lo largo del libro, nos referiremos a los objetos nombrándolos de acuerdo a su sintaxis exacta.

De acuerdo con esto la línea de código **document.documentElement.outerHTML**, mostraría íntegro todo el contenido de una página web (etiquetas incluidas), como puede verse en la Figura 18. De todas formas, la mayor parte de las veces no es necesario acceder al contenido íntegro de la página, resultando más útil acceder a porciones concretas del documento. Una vez más, no hay más que seguir el modelo de objetos para averiguar el procedimiento a seguir.

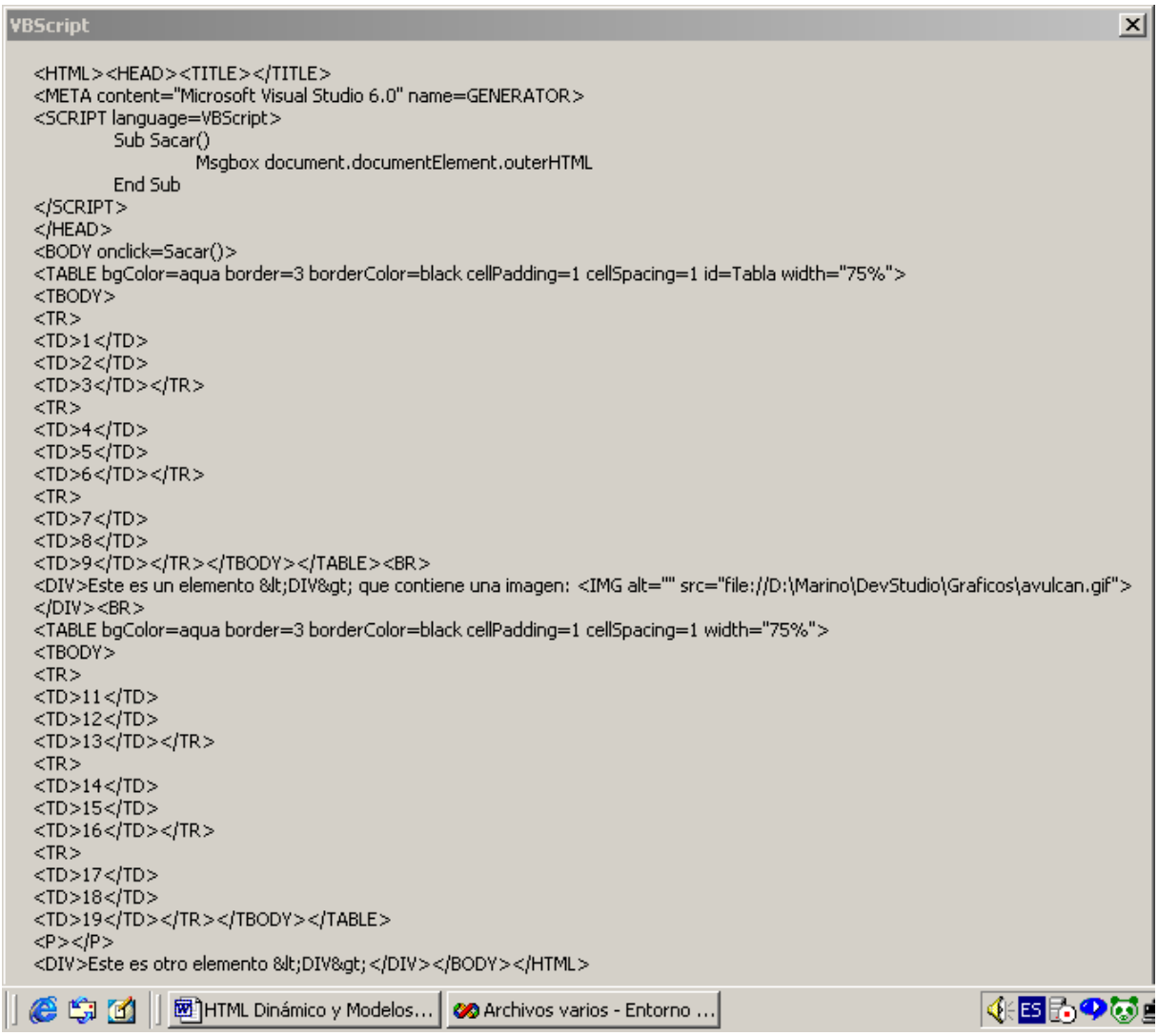

Figura 18. Caja de Diálogo mostrando el contenido completo del documento HTML mediante la propiedad **outerHTML** del objeto **documentElement**

Una vista jerárquica del modelo de objetos DOM

Al principio, puede ser útil para hacerse una idea del funcionamiento de DOM, crear gráficamente el modelo de objetos que se genera a partir de una página simple. El gráfico de la figura adjunta representa dicho esquema partiendo del documento que nos sirve de ejemplo. Nótese que en el esquema se representan todas las etiquetas como descendientes del propio documento HTML, representado aquí mediante el nodo o elemento **document**, que hace las veces de nodo raíz. Todos los otros nodos o elementos son descendientes de **document**, destacando, igualmente, **documentElement**, que corresponde con la marca <HTML>, que contiene al resto del documento.

Una vez creada la jerarquía de objetos de DOM para un documento dado, DOM expone al programador una serie de métodos pertenecientes a los objetos predeterminados que permiten el acceso a cualquier otro elemento de la jerarquía, siguiendo diferentes criterios de necesidad.

Por ejemplo, hasta ahora hemos visto como se accede a elementos basándonos en que éstos sean los elementos seleccionados por el usuario, pero podemos utilizar otros mecanismos de aproximación, como las colecciones predeterminadas o los métodos de acceso **GetElementsByID**, **GetElementsByTagName** y **GetElementsByName.**

En el caso de las colecciones predeterminadas el acceso estaba presente desde la primera implementación del modelo, mientras que los métodos citados derivan del modelo posterior de DOM.

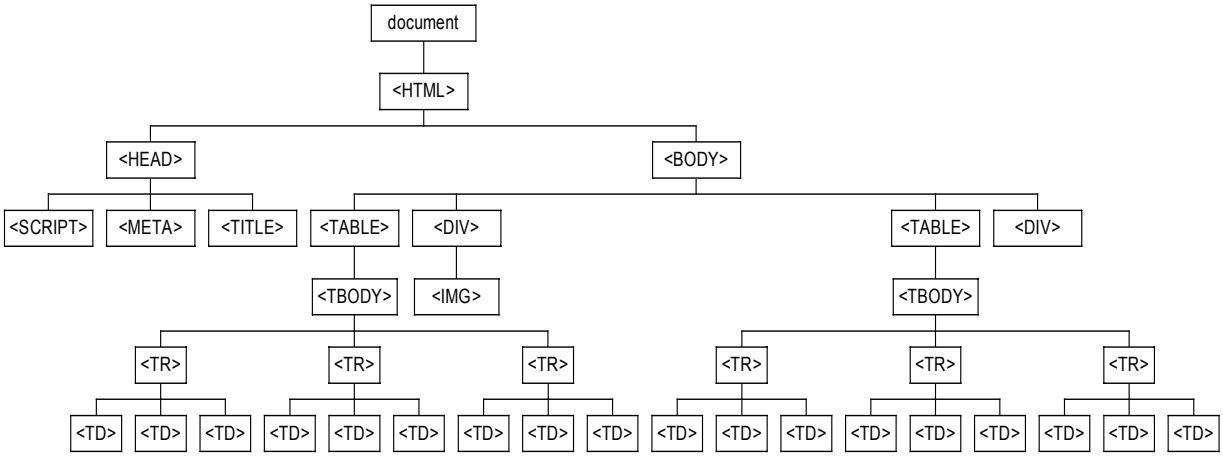

Figura 19. Modelo de objetos de DOM creado a partir de la página web de ejemplo

Supongamos, que queremos saber cuantas tablas tiene un documento. Siguiendo este razonamiento podemos obtener esos datos a partir de dos mecanismos distintos, el soportado por DHTML y el aportado por DOM mediante métodos.

En el primer caso, la colección **all**, del objeto **body** nos permite el acceso directo al dato a través de su propiedad **tags** mediante el Código fuente 19.

```
Msgbox document.body.all.tags("TABLE").length          'Devuelve un 2
```

Código fuente 19

Mientras que también lo podemos hacer a través de su propiedad **getElementsByTagName**, con una sintaxis similar a la que muestra el Código fuente 20.

```
Msgbox document.getElementsByTagName("Table").length   'Devuelve un 2, igualmente
```

Código fuente 20

Tenemos, pues, una jerarquía híbrida de cara al programador, que permite acceder a los elementos mediante varios métodos. Aquí nos centraremos, sobre todo, en el modelo de objetos de documento (DOM), que constituye una evolución consistente del modelo DHTML, y que se encuentra disponible en la versión de Internet Explorer 5.0 y posteriores.

Parece sólito comentar aquí las ventajas del modelo DOM respecto al anterior. Entre otras cosas, un programador puede:

- Mover una parte del árbol de documento a otra sin destruir ni recrear el contenido.
- Crear elementos nuevos y asignarles a cualquier parte del árbol de documento.
- Organizar y manipular ramas nuevas o ya existentes, antes de insertar nuevos objetos en la estructura.
- Más todas las posibilidades añadidas al modelo de Hojas de Estilo en Cascada (CSS1 y CSS2).

Al conjunto de propiedades y métodos específicos de DOM, se le conoce como **Interfaz de DOM**. En la Tabla 3 se listan algunos de los más importantes.

Propiedades	
data	Especifica el valor de un TextNode.
firstChild	Devuelve el primer nodo hijo de la colección **childNodes**
lastChild	Devuelve el último nodo hijo de la colección **childNodes**
nextSibling	Devuelve una referencia al siguiente hijo del nodo padre para el objeto especificado.
nodeName	Devuelve el nombre del elemento, similar a la propiedad **tagName**
nodeType	Devuelve el tipo de nodo: element, nodo texto, o atributo.
nodeValue	Devuelve el valor de un nodo.
parentNode	Devuelve una referencia al nodo padre.
previousSibling	Devuelve una referencia al hijo anterior del nodo padre para el objeto especificado.

specified	Devuelve un dato *Boolean* indicando si el valor de un atributo se ha establecido.
Métodos	
appendChild	Añade un elemento como hijo del objeto especificado.
applyElement	Convierte un elemento en hijo del objeto que se pasa como argumento a la función.
clearAttributes	Borra todos los atributos y valores del objeto especificado.
cloneNode	Copia una referencia al objeto partiendo de la jerarquía del documento.
createElement	Crea una nueva instancia del elemento especificado
createTextNode	Crea un nuevo nodo de texto a partir de la cadena indicada.
hasChildNodes	Devuelve un dato Boolean indicando si el objeto tiene nodos hijos.
insertBefore	Inserta un objeto en la jerarquía del documento.
mergeAttributes	Copia todos los atributos de lectura / escritura al objeto especificado
removeNode	Borra un nodo de la jerarquía del documento.
replaceNode	Intercambia un objeto existente con otro elemento.
swapNode	Intercambia la ubicación de dos objetos en la jerarquía del documento.
Colecciones	
attributes	Devuelve la colección de atributos para un objeto dado.
childNodes	Devuelve la colección de nodos hijos para un objeto dado.

Tabla 3. Listado de las propiedades y métodos principales de DOM

Se ha comentado que este conjunto de extensiones del primitivo DHTML permite, por ejemplo, la creación de elementos dinámicos (en tiempo de ejecución) como respuesta a eventos. Pongamos un ejemplo basado en la construcción de una tabla nueva a partir de nuestra página web básica utilizando ambos modelos, para ver las diferencias. Para evitar problemas en la creación de tabla, debemos utilizar lo que se denomina *HTML bien formado*, es decir aquel que sigue –lo más estrictamente posible- las especificaciones de HMTL tal y como se recomiendan por la W3C.

Ejemplo de Creación Dinámica de Tablas mediante el modelo DHTML

En el caso de las tablas, esto significa que debe de contener un elemento <TBODY> dentro del elemento <TABLE>. El Código fuente 21 crea una tabla nueva al final de documento, conteniendo 3 celdas cuyo contenido son los textos “Celda 1”, “Celda 2” y “Celda 3”, utilizando el modelo DHTML:

```
Dim sTable
sTable ="<TABLE Border='1' ID='Tabla3'></TABLE>"
document.body.innerHTML=document.body.innerHTML & sTable
Tabla3.insertRow(oTable1.rows.length)
Tabla3.rows(0).insertCell(oTable1.rows(0).cells.length)
Tabla3.rows(0).insertCell(oTable1.rows(0).cells.length)
Tabla3.rows(0).insertCell(oTable1.rows(0).cells.length)
Tabla3.rows(0).cells(0).innerHTML="Celda 1"
Tabla3.rows(0).cells(1).innerHTML="Celda 2"
Tabla3.rows(0).cells(2).innerHTML="Celda 3"
```

Código fuente 21

Analicemos la secuencia de comandos de VBScript:

En la variable sTable, se asigna primeramente la definición básica de la tabla y se añade al contenido del documento (propiedad *innerHTML*). Desde ese momento, el objeto tabla puede ser referenciado mediante su nombre: *Tabla3.* Utilizando los métodos *insertRow* e *insertCell,* añadimos a la estructura las correspondientes marcas <TR> y <TD>, mientras que las colecciones *rows* y *cells* nos dan acceso al contenido de las filas y celdas y a sus valores dinámicos (*length* nos indica el número de celdas, etc).

Si asignamos este código al evento Load del documento (puede ser cualquier otro), cuando visualicemos la página, veremos que en la parte inferior aparece la nueva tabla (Tabla 3), tal y como ha sido definida, con la particularidad de que DOM ha construido para nosotros un elemento <TBODY> que no habíamos creado expresamente, ya que al solicitar el código fuente mediante una caja de mensaje tal y como vemos en el Código fuente 22.

```
Msgbox oTable1.outerHTML
```

Código fuente 22

Obtenemos la ventana de la Figura 20, donde se ven los elementos <TBODY> enmarcando al resto de la tabla creada.

Figura 20. Tabla dinámica creada mediante código mostrando la presencia de elementos <TBODY>

Sin embargo, para crear una tabla dinámica mediante DOM, debemos crear explícitamente los elementos <TBODY>, si no queremos que los resultados sean impredecibles. El código correspondiente al mismo procedimiento mediante DOM sería el que muestra el Código fuente 23.

```
Dim oTabla, oTBody, oFila, oCelda, oCelda2, oCelda3
Set oTabla =document.createElement("TABLE")
Set oTBody =document.createElement("TBODY")
Set oFila =document.createElement("TR")
Set oCelda =document.createElement("TD")
Set oCelda2=oCelda.cloneNode() 'Crea una nueva celda a partir de la anterior
Set oCelda3=oCelda.cloneNode() 'Crea una nueva celda a partir de la anterior
document.body.appendChild(oTabla)
oTabla.appendChild(oTBody)
oTBody.appendChild(oFila)
oFila.appendChild(oCelda)
oFila.appendChild(oCelda2)
oFila.appendChild(oCelda3)
oCelda.innerHTML="Celda 1"
oCelda2.innerHTML="Celda 2"
oCelda3.innerHTML="Celda 3"
```

Código fuente 23

Podemos ver que el mecanismo utilizado aquí es ligeramente diferente, ya que los elementos se crean invocando los nombres estándar de cada objeto por su definición HTML: TABLE, TBODY, TR y TD. Una vez creados, es preciso asignarlos un lugar dentro del árbol jerárquico, lo que se consigue mediante el método *appendChild* (añadir nodo hijo), siguiendo el orden natural de la estructura de la tabla: primero se añade la tabla al objeto *body* del documento, después el objeto *oTBody* a la tabla, a continuación se le añade el objeto oFila, y finalmente, se añaden las dos celdas a la fila. Para concluir, como las celdas están vacías por defecto, se les asigna un valor.

Sin embargo hay un cambio más respecto al modo de trabajo anterior: no hemos asignado ningún formato a la tabla ni a las filas, por lo que el borde de la tabla será 0, y sólo se visualizarán los contenidos ("Celda 1", "Celda 2" y "Celda 3"), sin ver la tabla que los contiene. Si queremos repetir la situación que hemos creado antes, dando un borde a la tabla y asignándole un color de fondo, tenemos que crear esos atributos explícitamente, o bien copiarlos a partir de los atributos asignados a otro objeto similar.

Por ejemplo podríamos hacer que nuestra tabla oTabla, heredara los atributos de la primera tabla utilizando el método *mergeAttributes*, que tienen casi todos los objetos. Bastaría una sola línea de código para que la tabla tuviera borde igual a 1, y color azul.

```
oTabla.mergeAttributes(Tabla)
```

Código fuente 24

Esto significa que cualquier elemento puede ser destruido o creado a partir de código fuente y como respuesta a un evento de usuario y/o sistema.

Otras novedades que aporta DHTML a través de ejemplos

Vamos a ver algunos ejemplos de funcionamiento de las posibilidades de *Posicionamiento Dinámico, Filtros, Descarga de Tipos de Letra (Font Download)* y *Enlaces de datos.*

Posicionamiento Dinámico

Entendemos por posicionamiento dinámico la capacidad de situar un elemento en cualquier posición de la página dependiendo de unas coordenadas, y no de la situación relativa del elemento respecto al resto de elementos de la página. Esta funcionalidad se adquiere mediante los estilos, y concretamente utilizando los atributos ***position, top, left*** y ***z-index*** a los valores de estilo que se deseen.

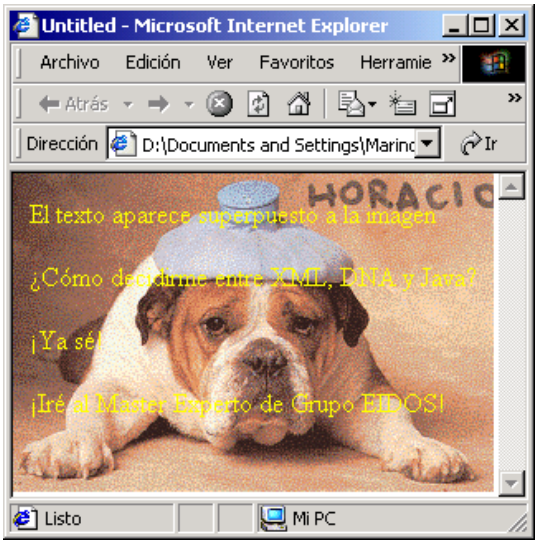

Figura 21. Página mostrando un texto superpuesto a la imagen

Por ejemplo, si queremos que un elemento ocupe una posición absoluta en unas coordenadas de pantalla, declaramos el valor de estilo **position** como **absolute,** y usamos **top** y **left** para indicar las coordenadas. Si además de situar el elemento en una posición concreta queremos que se sitúe por detrás o por delante de los otros elementos con los que pueda compartir una ubicación, daremos un valor a **z-index**, para establecer el nivel de profundidad en el teórico eje de las Z (el que mira hacia el espectador) dependiendo de la capa que queremos que ocupe.

En el ejemplo hemos situado la imagen "Horacio.gif" por detrás del texto de la página, indicando los valores que se muestran en el Código fuente 25 en el atributo **style** de la etiqueta <IMG>:

```
<body>
<font color="Yellow">
<p>El texto aparece superpuesto a la imagen</p>
<p>¿Cómo decidirme entre XML, DNA y Java?</p>
<P>¡Ya sé!</P>
<P>¡Iré al Master Experto de Grupo EIDOS!</p>
</font>
<img style="position: absolute; top: 0; left: 0; z-index: -1;"
src="../../Horacio.gif" width="291" height="192" border="0" alt="">

</body>
</html>
```

Código fuente 25

Además de la ubicación absoluta y de las coordenadas, el valor –1 de ***z-index*** indica al navegador que el objeto debe situarse en el nivel más profundo.

Filtros y Transiciones

Se trata de efectos visuales multimedia que pueden implementarse mediante propiedades de las Hojas de Estilo en Cascada. Existe una lista de filtros disponibles, pero de momento sólo valen para la versión de Internet Explorer 5.5. Su funcionamiento se basa en asociar a una etiqueta uno de esos filtros utilizando el atributo *filter*. En el estado actual de las cosas, se considera un elemento añadido a la última versión del navegador, pero no tiene la categoría de estándar.

Descarga dinámica de tipos de letra (Font download)

Cuando una etiqueta contiene el atributo de estilo ***@font-face*** el tipo correspondiente se descarga, se instala automáticamente, y se descarta una vez que la página es abandonada por el navegador. Observe el lector éste comportamiento, a través del Código fuente 26, tomado de la ayuda referente a DHTML en el MSDN de Microsoft.

```
<HTML><HEAD>
<STYLE>@font-face {font-family:comic;0
src:url(http://abc.domain.com/fonts/comicbold.eot);}
</STYLE>
</HEAD>
<BODY>
<p style="font-family:comic;font-size:18pt">this line uses the @font-face style
element to display this text using the Comic Sans MS font in 18-point size and
bold.
<p>
</BODY></HTML>
```

Código fuente 26

En el caso de que la letra indicada no exista, el navegador conecta con la página http://abc.domain.com/fonts/comicbold.eot para descargarla y mostrar su contenido formateado.

Datos Enlazados

El problema del tratamiento de datos, también se abordó en la definición del estándar HTML 4.0. Y una de las posibilidades que se sugirieron fue la de que los navegadores permitieran el enlace de datos a algunas etiquetas mediante algún mecanismo que permitiera mantener en memoria el equivalente a un Recordset.

Operando de esta forma, cuando se carga el documento, se produce inmediatamente la conexión y lectura de los datos enlazados. Un caso típico es el de un listado formateado mediante una tabla. La etiqueta <TABLE> puede ser enlazada a un origen de datos con este propósito y se genera automáticamente una fila en la tabla, por cada fila del conjunto de registros leído.

Una vez leída toda la información, es incluso posible realizar operaciones con los datos, como filtrarlos por una condición o establecer criterios de ordenación, sin que se requieran posteriores consultas al servidor. Otro uso común es el de presentar los datos enlazados a etiquetas del tipo <INPUT> y permitir la navegación por los registros, incluso admitiendo que el usuario realice cambios en ellos, y los envíe posteriormente al servidor.

Naturalmente, existe un elemento encargado de esta tarea de comunicación con el servidor y es el objeto ***Data Source Object***. Se trata de un control ActiveX o un *applet* de Java que enlaza con el origen de datos. A partir de la versión 4.0 de Internet Explorer, Microsoft incluyó dos objetos para los enlaces de datos, dependiendo de si se trataba orígenes de datos en formato delimitado por comas, u orígenes de datos tipo ODBC (SQL-Server, Oracle, etc.).

Modelo de funcionamiento de los objetos de enlace de datos.

A continuación se muestra un gráfico con el esquema de funcionamiento de un objeto DSO. El objeto DSO crea un caché en la máquina del cliente para almacenar los datos y mantiene un puntero de registro con el registro activo. Utiliza los mecanismos provistos por el estándar OLE-DB para el acceso a los datos, y mediante un *Agente de Enlace de datos (DataBinding Agent)* y un *Repetidor de datos en Tablas* (dos de las categorías implementadas por el componente DSO), se encargarán de preparar dinámicamente la información.

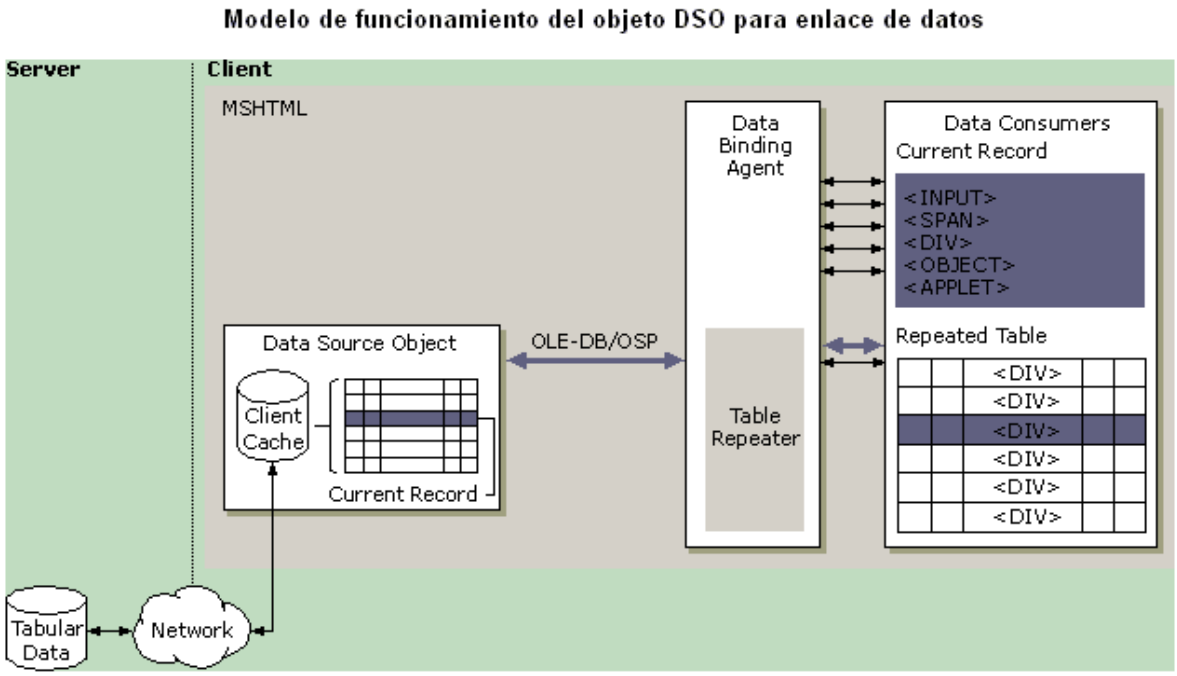

Figura 22. Un esquema de uso de un DSO según el MSDN de Microsoft

Ejemplo de uso de un Data Source Object (DSO)

Aunque una página web puede hacer referencia a este objeto de varias formas (mediante los distintos lenguajes de ***script***, cada uno con su sintaxis, y mediante etiquetas), vamos a usar un ejemplo en el que haremos referencia al DSO mediante etiquetas. Cuando se enlaza un objeto así, es preciso conocer el identificador de clase de sistema (el valor de su **CLSID**) que distingue a cada componente ActiveX, o disponer de un editor HTML que permita incrustar estos objetos mediante interfaces visuales y que realice por nosotros esa labor.

Para aclarar siquiera superficialmente el funcionamiento de esta tecnología a los lectores no muy avezados vamos a incluir una breve explicación del mecanismo operativo: cuando una herramienta de desarrollo crea un componente ActiveX (EXE o DLL), puede convertirlo en un servicio del sistema operativo registrándolo. El proceso de registro consiste en asignar a ese componente un identificador

único y anotar esos datos y otros complementarios en el registro de Windows para que cualquier aplicación que lo requiera pueda utilizarlo. Los componentes, reciben –de hecho- más de un identificador, pero los dos más importantes son su **CLSID** (un número de 128 bits, que no puede repetirse nunca) y su **ProgID**, una cadena de caracteres mediante la que se puede hacer referencia al componente desde entornos de desarrollo y producción, sin tener que recordar o buscar el número del CLSID.

En al documentación se nos indica que el número de CLSID del componente de datos es el valor hexadecimal {333C7BC4-460F-11D0-BC04-0080C7055A83}. Si entonces abrimos el editor del Registro del Sistema y buscamos ese número, aparecerá una entrada como la que vemos en la Figura 23.

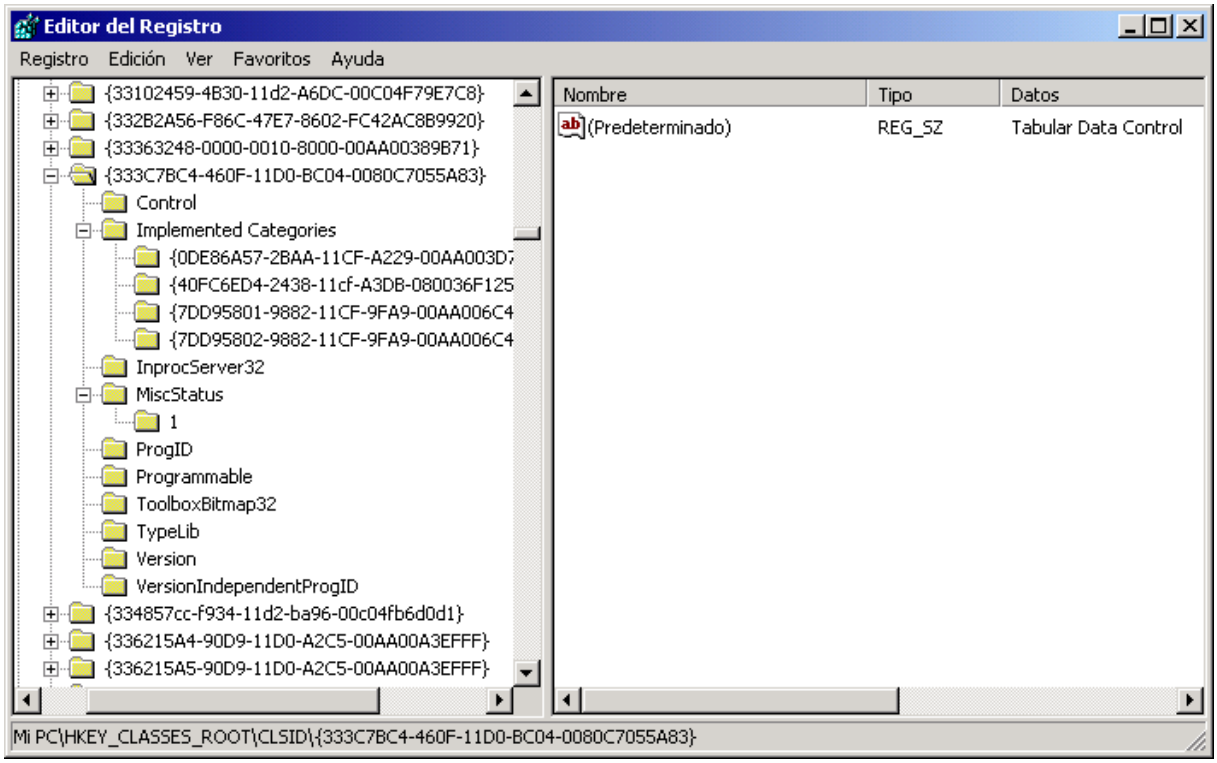

Figura 23. Registro de Windows mostrando el número de CLSID del control Tabular Data Control (conocido como DSO)

Observamos que el nombre con que se registra el DSO en el sistema es –en realidad- *Tabular Data Control*- (Control de datos de tipo tabular), y que tiene varias *categorías* registradas, que son los subcomponentes operativos de los que hablábamos antes en el esquema de funcionamiento.

Si queremos conocer más acerca de los métodos, propiedades y eventos que soporta este objeto (lo que denominamos su ***interfaz***), cualquier herramienta que pueda hacer referencia a él desde un entorno de desarrollo nos mostrará ese contenido, gracias a que en el momento de la generación del fichero físico de ese componente, se añaden las interfaces estándar **IUnknown** e **IDispatch** que son paquetes de funciones predeterminadas (construidas de acuerdo con el estándar DCOM) y pensadas para ofrecer al programador que los maneja una especie de *menú* del componente (la lista de todo lo que el componente es capaz de hacer).

Por ejemplo, podemos usar Visual Basic o una herramienta de Office para visualizar una interfaz utilizando el Examinador de Objetos, una vez que hemos referenciado el Tabular Data Control. Incluso desde Visual Interdev, es posible crear en la ventana de código una referencia al objeto utilizando su ProgID y automáticamente (mediante vinculación tardía), la herramienta utilizará la

interfaz ***IDispatch*** para mostrarnos el conjunto de propiedades cuando el proceso de desarrollo lo requiera. En este caso, leeremos del registro la cadena que corresponde a su ProgID de objeto, que es: **TDCCtl.TDCCtl.1**. Una vez que sabemos este dato, el siguiente código fuente editado sobre Visual Interdev nos muestra la interfaz, como se ve en la figura adjunta:

Esta sería una de las formas de acceder a un componente mediante código de *script.* El otro método de acceso consiste en hacer una referencia al CLSID del objeto usando una etiqueta <OBJECT>, que admite un atributo CLSID, y la posibilidad de asignación de parámetros adicionales mediante etiquetas <PARAM> contenidas en ella; en realidad, tantas como parámetros queramos asignar.

```
<HTML>
<HEAD>
<TITLE></TITLE>
<SCRIPT LANGUAGE=JavaScript>
<!--

function CrearEnlaceDatos() {
var x = new ActiveXObject("TDCCtl.TDCCtl.1")
x.
}
//
</
</
<B                    avascript onload="return CrearEnlaceDatos(
<P
</
</HTML>
```

AppendDa
CaseSensitive
CharSet
DataURL
EscapeChar
FieldDelim
Filter
Language
ReadyState
Reset

Figura 24. Ventana de código de Visual Interdev mostrando la interfaz del objeto Tabular Data Control

Veamos cómo funciona a través de un ejemplo: un fichero de texto delimitado por comas. Hemos preparado un fichero llamado *Direcciones.txt* que contiene una cabecera con campos, y datos separados por comas, en columnas delimitadas por un retorno de carro.

```
Nombre:String,Apellidos:String,E_Mail:String
Leoncio,León,lleon@cartoons.com
Juancho,Lagarto,ljuancho@cartoons.com
Magilla,Gorila,gmagilla@cartoons.com
Silvestre,Gato,gsilvestre@cartoons.com
```

Código fuente 27

El fichero contiene tres campos: Nombre, Apellidos e E_Mail y cuatro registros de datos. Para hacer referencia a ese fichero, utilizaremos el Código fuente 28.

```
<HTML>
<HEAD>
<TITLE>Acceso a Datos de un fichero separado por comas</TITLE>
```

```
</HEAD>
<BODY>
<Object ID="OrigenDeDatos" width=0 height=0
  Classid="clsid:333C7BC4-460F-11D0-BC04-0080C7055A83">
  <Param Name="DataURL" Value="Direcciones.txt">
  <Param Name="UseHeader" Value="true">
</Object>

<TABLE BORDER=1 bgcolor=Lime DATASRC="#OrigenDeDatos">
<THEAD>
<TR><TH>Nombre<TH>Apellidos<TH>E_Mail</TR> </TR>
<TBODY>
<TR>
<TD><SPAN DATAFLD=Nombre></SPAN></TD>
<TD><SPAN DATAFLD=Apellidos></SPAN></TD>
<TD><SPAN DATAFLD=E_Mail></SPAN></TD></TR></TBODY></TABLE>
</BODY>
</HTML>
```

Código fuente 28

Al visualizar la página en el explorador, obtenemos la Figura 25.

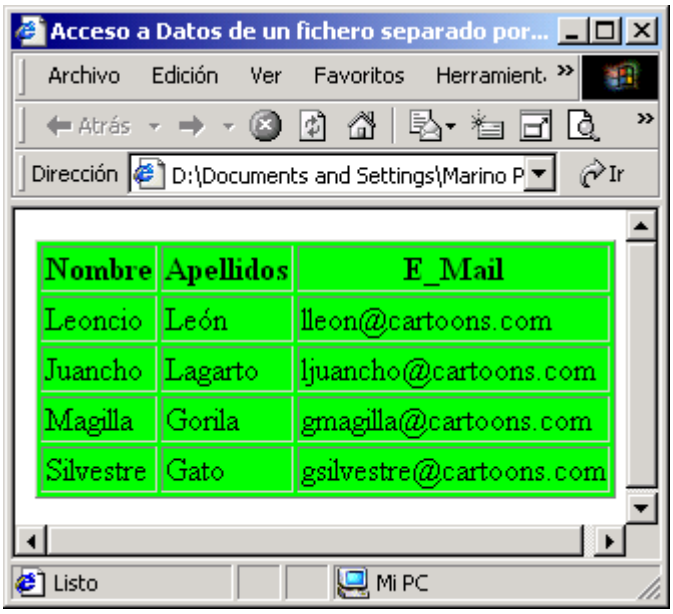

Figura 25. El componente DSO mostrando en una tabla el contenido del fichero Direcciones.txt

Además de ésta posibilidad se puede hacer referencia a un origen de datos remoto utilizando otra de las interfaces implementadas por el DSO, que hace referencia a un origen de datos remoto. En ese caso, las propiedades del componente cambian notablemente (cada interfaz implementada, contiene su propio conjunto de propiedades, métodos y eventos). No vamos a abordar esa situación, ya que va más allá del alcance de esta obra. Remitimos al lector a los siguientes eBooks: *"Acceso a Bases de Datos con Java-JDBC 2.0"*, *"Programación de Aplicaciones para Internet con ASP 3" y XML: Introducción al lenguaje"*, disponibles en la ***Librería Digital***.

A continuación, para acabar éste capítulo, vamos a hacer un análisis del soporte que ofrecen los navegadores respecto a las etiquetas definidas por el estándar HTML 4.0. Como podrá apreciar el lector la situación era bastante caótica hacia el momento de la aparición de las versiones 4.x de los navegadores, si bien la progresiva adaptación a los estándares está solucionando este problema.

HTML 4.0: Características e implementación en Explorer y NetScape

Vamos a revisar aquí la forma en que ambas implementaciones interpretan el lenguaje HTML en su versión 4.0. Las versiones posteriores han ido mejorando su nivel de soporte, como ya hemos dicho, pero todavía puede afirmarse que –a la fecha actual, **Enero/2001**-, más de la mitad del parque de navegadores utiliza versiones 4.x de los navegadores.

Lo primero que debiera de decirse sobre el lenguaje (también sobre la anterior implementación, la HTML 3.2), es que la especificación completa de funcionamiento es un documento enorme y complejo, que analizado con profundidad sorprende por la gran cantidad de características que posee. Y para mejorar las cosas, las herramientas de creación de páginas son –algunas veces- realmente pobres, o codifican un HTML incorrecto. Esto nos lleva a otra alternativa: crear las páginas con un editor de texto. Pero, también en este caso nos llevaremos sorpresas al visualizarlo si, simplemente, seguimos el estándar. Y ello debido a la interpretación del navegador. Pero veamos primero lo que entendemos por estándar.

Estándares e implementaciones

Comencemos estableciendo que NO existe un estándar real para HTML. O, al menos, no en la forma de un grupo oficial de estandarización como el Instituto ISO (hay un proyecto actual de estandarización ISO-HTML, pero se encuentra en pañales y no vamos a tratarlo aquí). Lo que sí existe es el arriba citado W3C que funciona como un estándar de facto, con miembros de las principales compañías productoras de software para Internet, estableciendo recomendaciones, que no estándares. Netscape y Microsoft son miembros destacados, por lo que resulta incorrecto afirmar que las recomendaciones no reflejan las opiniones de los fabricantes de estos navegadores. Incluso existen listas abiertas de correo, donde cualquiera puede contribuir a la discusión y desarrollo de nuevas especificaciones. En ese sentido W3C es una organización muy abierta.[2]

Con esto en mente, sorprende observar las diferencias de implementación entre los dos navegadores y, entre estos, con el estándar. Las razones: imagino que serán de pura mercadotecnia, en principio. Lo cierto es que Netscape, que proviene del viejo Mosaic, fue el primero en mejorar las prestaciones permitiendo que el código HTML pudiese ofrecer prestaciones que fueron haciéndose populares, iniciando así el efecto bola de nieve que le llevó a tener una cuota de mercado incomparable. Además, Netscape estaba disponible para una gran variedad de plataformas, y las recomendaciones del W3C tenían poco eco posterior.

Pero cuando Microsoft entró en escena, las cosas cambiaron. Al principio, el Explorer se parecía mucho al Navigator, y con buen criterio, ya que así se conseguía que los usuarios perdiesen el miedo al cambio. También introdujeron nuevas características, que al igual que sucedió con Navigator, tuvieron una desigual acogida. A medida que iban creciendo en prestaciones y novedades empezaron a separarse, y, de aquellas lluvias, vinieron estos lodos en los que nos encontramos hoy.

Volvamos a las especificaciones. La que nos ocupa, es, curiosamente, la tercera, puesto que lo primero que pudo llamarse especificación fue un intento del W3C de poner orden en la situación 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.

---

[2] Vea el lector el comienzo de este manual para más datos acerca de la W3C

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 de 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 que es, hoy por hoy, la más estable de las implementaciones y la primera que puede llamarse estándar de facto.

Por su parte, HTML 4.0 es la última y también la mejor de las especificaciones y promete resolver muchos de los problemas que se presentan hoy en día, extendiendo las capacidades del lenguaje en muchas áreas y añadiendo posibilidades más acordes con las necesidades de mercado. Sin embargo, probablemente, será el último, tras una larga agonía. La razón, es que su sucesor, XML resuelve muchos de los problemas insolubles por HTML 3.2, prometiendo un auténtico lenguaje con elementos multimedia, tratamiento de datos, etc. Pero eso, será otra historia...

SGML y HTML

Según S.Piperoglou (1998), la implementación que se hace de las marcas en ambos navegadores no es correcta. En teoría, –y en aquellos lugares donde lo permite la implementación del estándar- omitir el final (o el comienzo) de una marca no debiera tener mayor impacto en la visibilidad del documento. Sin embargo, en muchas ocasiones, ambos navegadores rechazan asumir implícitamente la marca complementaria. Podemos entender, no obstante, que incluir siempre ambas marcas se trata de una buena práctica, si consideramos que el modelo de objetos de los dos navegadores rechazará aquellos elementos que no se encuentren correctamente completados. Ambos, igualmente, crearan objetos para cada marca que encuentren sin preocuparse por sus categorías.

Esto sucede en las listas en las que se incluyen datos, en lugar de elementos (ítem). El soporte parcial que incluyen Explorer y Netscape de las Hojas de Estilo en Cascada (CSS1: Cascade Style Sheets), hace que estas aproximaciones sean, ahora, innecesarias. Siempre será preferible utilizar las herramientas para aquello para lo que fueron diseñadas.

Comentarios, atributos y caracteres especiales

Respecto a los atributos, la norma a seguir debiera ser la de escribir entre comillas cualquier valor. Los navegadores ya citados no tienen problemas generalmente para entender el significado aunque esto no se haga así, pero el resto de navegadores no. Tampoco debieran utilizarse los nombre largos en atributos booleanos.

Algo similar sucede en las referencias a caracteres especiales, (<,>,ñ, etc.) que siempre debieran de terminarse con un punto y coma, aunque los navegadores puedan interpretarlos sin necesidad de su inclusión. Tanto Netscape como Explorer reaccionan de forma extraña con las referencias a caracteres UNICODE que no pueden interpretar, mostrando casi siempre los signos (&) y (;) en pantalla.

Finalmente, respecto a los comentarios de código, debe de tenerse en cuenta que los navegadores pueden interpretar erróneamente los mismos dando por terminado el comentario que comienza por (<!--) cuando encuentran el primer final de marca (>), en lugar de su correspondiente símbolo (-->). Al contrario de lo que se indica en la especificación, los navegadores no rechazan la presencia de dos o más guiones dentro de un comentario.

Tipos de datos

Ambos navegadores se adhieren bastante bien al estándar a este respecto, con unas pocas excepciones:

URLs

Los caracteres especiales, y en concreto el famoso ALT-126 (~), debieran ser convertidos a sus secuencias de escape correspondientes. Así, la dirección www.direccion.com/~ventas debiera de convertirse en www.direccion.com/%7eventas para garantizar su interpretación independiente del navegador.

Colores

Aquí también se ofrece una funcionalidad extendida. Los dos navegadores admiten más colores que los 16 estándares. Aquí una vez más se desaconseja la utilización de los nombres predefinidos en favor de su notación hexadecimal, carente de ambigüedad, aunque menos legible. De todos modos, aquí también se recomienda la utilización de hojas de estilo, que permiten una utilización más racional de los colores dentro de un contexto.

Multilongitud

Los dos navegadores soportan sólo parcialmente esta característica (la capacidad de indicar longitudes genéricas mediante asteriscos en la forma: n*). Puede utilizarse en los elementos colgroup y frameset donde ambos interpretan un asterisco simple como 1* acorde con la especificación.

Tipos de Hiperenlaces

Desgraciadamente, el único tipo de enlace soportado es el StyleSheet. Netscape también soporta el tipo de enlace Fontdef para vincular una fuente externa cuando se utilizan fuentes dinámicas, pero de eso hablaremos en otro apartado.

Descriptores de Media

Sólo los soporta el Explorer, quien dispone de tres de ellos: screen, print y all. No soporta listas de descriptores separadas por comas, y el único efecto apreciable de una declaración similar es ignorar la presencia de una declaración media=print dentro de una stylesheet. Netscape no soporta los descriptores de media.

Nombres de marcos de destino

Ambos navegadores interpretan los atributos target como nombres de ventanas, y no como nombres de marcos, en caso de que el marco correspondiente no exista. Nunca se debe usar esta técnica para mostrar documentos en otras ventanas. Es preferible usar cualquier lenguaje de script para lanzar otra ventana, de la cual –además- se puede tener control total y posible reutilización posterior.

Declaraciones de Tipo de documento

Se entienden por tales las declaraciones básicas de HTML donde se especifica el tipo de documento y sus elementos estructurales, como por ejemplo <HTML>, <HEAD>, <BODY>, etc. El tándem Explorer-Navigator ignora tales declaraciones, haciendo por su cuenta todos los arreglos necesarios para la interpretación del documento. No obstante, y como norma es recomendable incluirlas por claridad y también por que otros navegadores no realizan esa labor.

La marca <HTML>

Puede omitirse ya que no tiene el más mínimo efecto en ningún navegador ni siquiera a efectos de "renderización" (interpretación del código HTML). El atributo versión no está soportado en ningún caso.

La marca <HEAD>

También puede omitirse con tranquilidad. El antiguo atributo profile ya no tiene ningún efecto.

La marca <TITLE>

Los contenidos del elemento <TITLE> se muestran en la barra de título de ambos navegadores e incluso la mayoría de las referencias a caracteres especiales funcionarán bien igualmente. Sin embargo, y como norma, deberíamos de ser cuidadosos al respecto ya que el titulo que incluyamos allí es el que aparecerá por defecto cuando el usuario agregue una marca a nuestra página en su explorador, por lo que deberá de ser suficientemente explicativa de su contenido. Respecto a los marcos, recordemos que ambos navegadores sólo muestran los del marco base (frameset), cosa que hace tal práctica desaconsejable según varios autores.

En caso de omisión, los navegadores mostrarán la dirección URL en la barra de título, aunque el estándar y la buena práctica de programación sugieren su inclusión en todos los casos.

El atributo <TITLE>

Tratado como atributo, el único navegador que lo reconoce es Explorer, quien muestra una etiqueta flotante (tooltip), asociada al elemento que lo contiene, cada vez que el cursor se desplaza sobre su posición en pantalla Puede ser útil para dotar de información adicional a los hipervínculos poco explicativos (como por favor pulse aquí).

La marca <META DATA>

Se trata de atributos de cabecera contemplados en el protocolo de transferencia de hipertexto HHTP, y no específicamente pertenecientes a HTML; de ahí su nombre de <META DATA>. Aunque no la soporta ninguno de los dos navegadores, puede ser muy útil para suministrar información a los buscadores y facilitar la visualización de los editores. Con tres notables excepciones:

a) El antes popular mecanismo Refresh que permite dirigir una página a otra al cabo de un tiempo determinado.

```
<meta http-equiv="Refresh" content="3;URL=http://www.compañia.com/ventas/">
```

Código fuente 29

Cargará la dirección URL http://www.compañia.com/ventas/ después de 3 segundos. Esto es poco recomendable, y la propia especificación HTML cita una forma diferente de hacerlo (que tampoco funciona). Es preferible conseguir el mismo efecto utilizando JavaScript.

b) Los dos navegadores también soportan la cabecera Expires tal y como aparece en el Código fuente 30.

```
<meta http-equiv="Expires" content="Fri, 9 Sep 1997 12:52:34">,
```

Código fuente 30

Para indicar al navegador la fecha a partir de la cual el documento no deberá de seguir siendo almacenado en el caché del equipo cliente. Si el valor fuese 0, la acción se produce inmediatamente.

c) También es posible establecer cookies mediante esta marca, pero no vamos a detenernos aquí en la técnica utilizada para ello. Debería utilizarse el servidor para establecer y controlar las cookies, fuera aparte de otras consideraciones de tipo ético.

La marca <BODY>

Existe aquí una coincidencia total en la forma de implementación con respecto al estándar. Tal y como se indica en éste último, el uso de elementos de presentación en esta marca debiera de evitarse, lo que significa que si no le importa que sus usuarios de Navigator 3 y Explorer 2, asuman sus valores por defecto, incluso podría eliminarse.

También hay que significar que ésta marca admite caracteres fuera de elementos que se sitúan al nivel de bloque, tales como los párrafos. Afortunadamente, tales caracteres serán ignorados por el constructor del modelo de objetos del documento, en lo que se refiere al tratamiento con lenguajes de script, o al manejo de Hojas de Estilo en Cascada.

Elementos identificadores

Ambos navegadores soportan los atributos ID y CLASS para el propósito con que fueron creados, que no es otro que el de identificar de forma unívoca a un elemento concreto dentro del modelo de objetos del documento, para que posteriormente pueda ser reconocido por el código de programación JavaScript o VBScript.

Bloques y elementos “en línea”

Los dos navegadores permiten anidamientos erróneos, y a menudo, permiten a un elemento incluir otro elemento al nivel de bloque, cuando debería ser un elemento en-línea. La recomendación es cuidar adecuadamente los anidamientos, incluir elementos en línea exclusivamente cuando el elemento no admita otros al nivel de bloque, tal y como aparece en la especificación.

Marcas de Agrupación

Las marcas <DIV> Y <SPAN>, son reconocidas por ambos navegadores. Recordemos que la utilización de <DIV> se centraba fundamentalmente en evitar el uso del atributo CENTER. Ahora, lógicamente, tras la aparición de las Hojas de Estilo su uso es muy común.

Encabezamientos

Los encabezamientos son, igualmente, interpretados de la misma forma, a pesar de que ambos admiten elementos al nivel de bloque en su interior, algo expresamente prohibido por el estándar y que puede evitarse con métodos alternativos. Por cierto, debido a la defectuosa implementación del tratamiento de cabeceras por parte de ambos navegadores, puede observarse una molesta tendencia a asumir sus valores por defecto, lo que, sumado al soporte incompleto de las Hojas de Estilo (CSS), hace que lo que –en principio- debiera de asumirse como una construcción lógica, termine por ser un mero examinador de texto en negrita. Deberían usarse para reconocer cabeceras, y dejar las Hojas de Estilo para aplicar formatos.

La marca <ADRESS>

Se interpreta como texto en cursiva en ambos navegadores y su funcionalidad equivale al elemento <i> en ambos navegadores. Aunque se permiten elementos al nivel de bloque en una marca <ADRESS> esto está, normalmente, prohibido.

El Texto: párrafos, líneas y frases

El espacio en blanco

Respecto a este elemento, se siguen las reglas escrupulosamente, con excepción de aquella que dice que un espacio en blanco inmediatamente después de una marca de inicio, o antes de una marca de fin, debiera de ignorarse. Su utilización puede provocar una interpretación poco afortunada y debe evitarse. El carácter más útil para incluir un especio en blanco, es el carácter de espacio de no-ruptura ( ). No obstante su corrección no es muy saludable su uso.

Frases

Los elementos frase, se interpretan como elementos de estilo de tipo de letra: em, cite, dfn y var se interpretan en cursiva, strong se convierte en negrita, y code, samp y kbd aparecen en tipo de letra monospaced (de espaciado único, como la Courier).

En cuanto a abbr y acronym, Netscape no las reconoce. Y Explorer reconoce la segunda, pero no hace nada con ella, aunque muestra una etiqueta flotante con el atributo título si éste se encuentra presente.

Citas

La marca <BLOCKQUOTE> se soporta y maneja de la misma forma que si se tratara de un texto sangrado. El Explorer también reconoce la marca <q>, pero no inserta las comillas para delimitar su contenido. Ninguno de los dos reconoce el atributo cite.

Párrafos

Los párrafos, y en particular la marca <p>, son una de las más claras inconsistencias de ambos navegadores. Esencialmente, ambos navegadores consideran una marca <p> como una doble línea de ruptura, aunque la cosa tiene su intríngulis.

Desde el punto de vista estricto, sucede lo siguiente: Las marcas se interpretan correctamente con las siguientes excepciones: si no van seguidas de ningún texto, se interpretan como un retorno de carro, en contra de lo que dice el estándar. Esto ha dado lugar a que muchos creadores de páginas las utilicen para crear líneas en blanco en detrimento del uso de Hojas de Estilo. Además, un espacio después de un párrafo y antes de un elemento al nivel de bloque no se interpreta si no se incluye la marca de fin de párrafo , un encabezamiento, una lista, una cita o una marca <pre>. Veamos un ejemplo en el Código fuente 31.

```
<p>Esto es un párrafo.
<hr>
<p>Esto es otro párrafo.
```

Código fuente 31

Esto se interpreta sin ningún espacio entre el primer elemento y la línea separadora, sin embargo, si añadimos aquí el fin de párrafo en la primera línea, se verá como existe una línea en blanco antes de la línea recta separadora. Como norma, los finales de párrafo deben añadirse siempre.

```
<p>Esto es un párrafo.</p>
<hr>
<p>Esto es otro párrafo.
```

Código fuente 32

Rupturas de línea

La marca <br> se interpreta tal y como se indica en el estándar, y el carácter especial   también se usa como una forma de inhibir las líneas de ruptura.

Guiones

Ninguno de los navegadores soporta el carácter de guión blando , mostrándolo como un guión normal, y rehusando dividir la línea cuando esto sucede.

Texto preformateado

La marca <pre> es soportada por ambos navegadores para el control de texto preformateado, si bien ninguno de los dos maneja el atributo width. Netscape, sin embargo, soporta el atributo cols que viene a ejercer una función similar, especificando el número de columnas en el elemento, así como el atributo lógico wrap, que indica que el texto debe dividirse al llegar a la anchura establecida como tope, o la de la pantalla si el atributo cols no está presente.

Cambios en el documento

Netscape no reconoce los elementos ins y del. Explorer si, y maneja el contenido de los elementos del como si fuera un tipo de letra tachada (strikethrough). No soporta el atributo cite y no realiza ninguna acción apreciable con el valor del atributo datetime.

Listas

Las listas se soportan por ambos navegadores exactamente en la forma en que se establece en el estándar, incluyendo los atributos obsoletos, a excepción del atributo compact que no es soportado en ningún caso.

Tablas y Netscape

Antes de hablar de las características del nuevo Modelo de Tablas HTML 4.0, conviene recalcar que Netscape no lo soporta en absoluto, reduciendo su funcionalidad al modelo HTML 3.2, que soporta en su integridad. De forma que si el lector desarrolla para Netscape únicamente, puede omitir el resto de éste apartado.

No obstante, Netscape soporta algunas características adicionales al modelo HTML 3.2. Así, se manejan los atributos hspace y vspace, que establecen la distancia en píxeles entre la tabla y el texto que la rodea. También reconoce los atributos background y bgcolor para las marcas <TABLE>, <TR>, <TH> y <TD> que definen, respectivamente, la imagen y el color de fondo del elemento. También se soporta el atributo lógico nowrap para las celdas de una tabla, tal y como sugiere el estándar HTML 4.0.

Téngase también presente que Netscape no puede realizar una interpretación incremental de tablas independientemente de la información que esté incluida en ellas. Explorer, por su parte si realiza este trabajo incremental, ya que formatea la información según la recibe.

La marca <TABLE>

Explorer reconoce todos los atributos de la marca <TABLE>, tal y como se especifican el estándar, a excepción del elemento <summary>. También admite otros tres atributos: bordercolor, bordercolorlight y bordercolordark, que establecen el color del borde principal, el borde resaltado, y la sombra del borde cuando se interpreta con efectos 3D en el Explorer. Lo que no soporta es el atributo dir para manejar la dirección de escritura de las celdas.

La marca <CAPTION>

Explorer la reconoce, aunque interpreta los valores left y right del atributo align como alineación del texto del título de la tabla (Caption), en vez de como la posición de éste en la tabla misma. También se reconoce el atributo valign con valores top y bottom para la alineación del título en la celda.

Columnas, filas y agrupaciones de ambas

Explorer soporta las marcas thead, tfoot, tbody, colgroup, col, tr, th y td de acuerdo con la especificación, con algunas excepciones: ninguno de los elementos citados puede alinearse estableciendo el atributo align a justify, y los contenidos no pueden ser alineados respecto a un carácter mediante aling=char y los atributos char y charoff. De igual forma, ignora el atributo axis, referente a las celdas de la tabla. Por lo demás se ajusta el estándar totalmente.

Hiperenlaces

La marca <A>

Los dos navegadores soportan su funcionalidad básica: como marcadores y como encabezamientos de hiperenlace. Sin embargo, los atributos hreflang, type, charset, rel y rev, siguen sin considerarse. El único atributo útil es title que sólo lo soporta Explorer, que muestra la correspondiente etiqueta flotante (tooltip), con su contenido, amén de usar el texto como valor de marcador de usuario por defecto.

La marca <LINK>

Debido a una cierta falta de interés en especificar tipos de enlace en el estándar anterior, ésta marca se soporta de forma superficial. De hecho, el único tipo de enlace soportado por ambos es rel=stylesheet, que, lógicamente, indica una Hoja de Estilo. Netscape, también soporta rel=fontdef, para indicar un fichero de tipo de letra si se quiere utilizar la característica Fuentes Dinámicas de Netscape. Así mismo, ambos navegadores soportan el atributo type, como forma de definir el lenguaje de la Hoja de Estilo a utilizar. (Como ya hemos citado anteriormente, Explorer soporta también el atributo media, cuyo único efecto consiste en ignorar las Hojas de Estilo enlazadas mediante el atributo media=print.

Objetos, Imágenes y Applets

La marca <IMG>

Excepción hecha del atributo longdesc (descripción larga), ambos navegadores soportan esta marca de acuerdo con el estándar, si bien –debido a la creciente importancia de los elementos multimedia- le han añadido algunas características por su cuenta.

Conclusión

Como puede ver el lector la situación respecto a las versiones 4.x de los navegadores era problemática. Afortunadamente, los sucesivos parches y *versiones de refresco*, han ido acercándose cada vez más a ese estándar de facto que es la recomendación de W3C HTML 4.01.

2

Programación de Hojas de Estilo en Cascada (CSS)

Introducción

Desde el comienzo de los procesadores de texto siempre ha existido la necesidad de formatear los documentos de acuerdo a un conjunto de normas que se establecen previamente de acuerdo a unos requisitos: documentos empresariales, orientados al marketing, generación de informes de resultados o análisis... etc. Dichos documentos establecen inicialmente unas propiedades que se van a mantener durante toda la etapa de generación del mismo. A este conjunto de propiedades que establecen la **forma** del documento, le llamaremos **formato**.

Tan importante es generar el contenido del documento como generar el **formato** del mismo. Pues, de echo, es la cara visible que cualquier persona va a tener accesible cuando quiera consultar los contenidos del documento. La importancia de la **forma** estriba en el nivel de satisfacción que deseamos que tenga la persona que va a consultar nuestros informes: los colores empleados, la forma de las letras, la colocación de los párrafos, la sencillez de búsqueda de información concreta... Son elementos que no se pueden tomar a la ligera y menos ignorar. Puesto que el futuro uso de nuevos documentos (o reutilización de los ya existentes) por parte del mismo usuario depende de ello.

Si estos conceptos ya son importantes en la vida ordinaria y en la documentación generada por diversas fuentes, con más razón en Internet, donde la forma muchas veces marca el éxito de las páginas Web o bien la caída en el olvido en los inmensos mares de bits de la Red mundial.

Pero en Internet surge una necesidad - que ya existía en los primeros sistemas de edición de documentos, pero que no está tan marcada - : la posibilidad de poder cambiar de manera sencilla y

rápida el formato de los documentos. ¿Por qué?... porque Internet es un medio totalmente dinámico, que siempre está cambiando, evolucionando, probando nuevos elementos visuales. Y las páginas Web mantienen una auténtica "guerra" a ver quién consigue un formato visual más atractivo, que enganche al usuario y así fidelizarlo lo más posible. Debido a esta cadena continua de cambios, los documentos HTML deberían proporcionar mecanismos que permitieran "cambiar" la forma del documento de una manera ágil, sin que afecte a la estructura del mismo, de tal manera que a base de asignar nuevas "plantillas" de formato, la **forma** del documento aparentara cambiar radicalmente de aspecto. Pudiendo ser depurado éste de manera continua, y dando, además, la posibilidad de verse mejorado a base de adición de nuevos elementos de formato. Se consigue así **independizar el contenido de la forma**.

A este conjunto de "plantillas" de formato de documentos las denominaremos **hojas de estilo**. Y serán las que determinen las propiedades graficas de formato que deben cumplir todos y cada uno de los elementos HTML de los que se compone una página Web. Puesto que podremos estructurar diversas *hojas de estilo* en función a diversos criterios, y dichas hojas aplicarlas sobre un mismo documento, ampliaremos el nombre las mismas por el de **hojas de estilo en cascada**, dando a entender que es posible combinar varias hojas de estilo para obtener efectos de presentación complejos, fruto de la aplicación *en cascada* de las mismas. Y si en algún momento el diseño no va de acorde a las expectativas, siempre será posible cambiar aquella hoja que no es de agrado y sustituirla por otra mejor sin afectar al resto de los componentes.

Las CSS o Cascade Style Sheets (hojas de estilo en cascada)

CSS (*Cascade Style Sheets, hojas de estilo en* cascada) es una recomendación basada en el estándar de Internet del W3C, que describe el conjunto de símbolos, reglas y asociaciones existentes entre los elementos del lenguaje que forman un documento HTML y su apariencia gráfica en pantalla: color, fondo, estilo de fuente, posición en el espacio...etc.

CSS establece los elementos necesarios para independizar el contenido de los documentos HTML de la forma con la que se presentan en pantalla en función del navegador empleado. CSS extiende las normas de formato del documento, sin afectar a la estructura del mismo. De esta manera, un error en un formato, un navegador que no soporta colores...o cualquier otro error relacionado con el formato del documento no invalida el mismo.

CSS además también aporta elementos dinámicos de formato, pues es posible capturar el estilo de los elementos cuando se produce una reacción sobre ellos. De esta manera es posible evitar la escritura de código JavaScript para conseguir efectos tan vistosos como el subrayado automático de hipervínculos, botones gráficos, fondos de color en controles input html... etc.

¿Es todo correcto en las CSS?

Aunque CSS es una norma que permite independizar de una manera extremadamente flexible el contenido de la forma, lo cierto es que también es una fuente potencial de errores de concepto.

Puede ocurrir que nos olvidemos de la máxima absoluta del diseño independiente de contenido – forma y diseñar los documentos orientados a las hojas de estilos que ya están creadas para poder obtener los efectos de tabulado deseados. Esto no es correcto, puesto que en un futuro modificar la forma implicará modificar el documento.

Así mismo ocurre también tener la tentación de modificar los elementos base de HTML para producir efectos de herencia y evitar así trabajo innecesario a la hora de rediseñar el documento. Pero es también un error, puesto que futuras aplicaciones de la hoja de estilo invalidaría el uso correcto de las normas de presentación elementales de HTML. Y más si un tercero va a diseñar páginas nuevas y no conoce la aplicación de las normas de estilo del documento, viéndose forzado a rescribir las reglas originales de nuevo.

Con lo cual, podemos comprobar que si no se realiza un buen diseño el empleo de CSS puede desembocar en un descontrol de formatos que incluso pueden machacarse unos a otros. Haciendo inviable la independencia contenido – forma.

Como resumen a este punto, dictar una nueva máxima absoluta: **diseñar los documentos sin estilos. De manera que puedan ser correctamente leídos en cualquier tipo de entorno. Un documento nunca dependerá de su estilo. Después, y cuando las normas de diseño lo exijan, será cuando apliquemos estas "mascaras" de formato gráfico que llamaremos "estilos"**.

Comenzando con las CSS

CSS establece el conjunto de símbolos y reglas que nos permiten indicarle a HTML el cómo mostrar el contenido. Se puede incorporar la referencia CSS en las hojas HTML de 3 formas:

a) CSS Internas

Establecen la definición de marcas CSS dentro del mismo documento HTML sobre el que hacen efecto. Su ámbito es el documento al que pertenecen. Puesto que son exclusivas del documento, no se puede reutilizar su formato a no ser que sean copiadas a otros documentos HTML.

a.1) in-line

Marcas CSS que se sitúan en el mismo tag HTML sobre el que queremos que se aplique el estilo de formato. Son las más específicas de la recomendación. Sólo afectan al elemento HTML que las incorpora. El resto de elementos HTML, que ya no incluyen dicha definición CSS asumen las propiedades por defecto del navegador. Se colocan dentro del cuerpo BODY del documento HTML.

```
<!—Este tag IMAGE asume el estilo de tamaño de 64x64→
 <IMG STYLE="height:64;width:64" SRC="icono.gif">
<!—Este tag IMAGE asume el tamaño predeterminado de l a imagen Icono2.gif→
 <IMG SRC="icono2.gif">
```

Código fuente 33

a.2) *Bloque*

Bloques de definiciones CSS dentro de la misma página HTML sobre las que se aplican. Se determinan dentro del bloque HTML **<STYLE></STYLE>** y deben ir colocadas entre los tags de cabecera del documento - Desde HTML hasta BODY -. Su definición se hace extensible a todos los elementos HTML coincidentes de la página. La apertura de nuevas páginas no hereda las definiciones de la página actual.

Un ejemplo de un estilo en bloque se vería en el Código fuente 34.

```
<HTML>
<HEAD>
<STYLE TYPE="text/css">
      H1{font: 15pt "Arial";color: blue}
      P {color: red; text-decoration: italic}
</STYLE>
</HEAD>
      <BODY>
             <H1>Cabecera con estilo</H1>
             <P>Párrafo con estilo</P>
             Texto sin estilo
      </BODY>
      </HTML>
```

Código fuente 34

b) CSS Externa

Presentan el hipervínculo a un documento que en su interior almacena definiciones CSS. Se aplicarán a todas las páginas que tengan la referencia a dicho documento. La definición del enlace debe hacerse entre los tags HTML **<HEAD></HEAD>.**

```
Estilo.css

BODY
{
    BACKGROUND-IMAGE: url(/images/fondo.gif);
    BACKGROUND-REPEAT: no-repeat
}
H1
{
    font: 15pt "Arial";
    color: blue
}
```

```
Mipagina.htm
<HTML><HEAD>
<LINK REL=STYLESHEET HREF=”Estilo.css” TYPE=”Text/Css”>
</HEAD>
<BODY>
<H1>Titulo con estilo</H1>   Prueba de documento con estilo e imagen de fondo
</BODY></HTML>
```

Código fuente 35

Este tipo de referencias son el equivalente a las “templates” o plantillas de procesadores de texto como Microsoft Word. Permiten que los sitios Web mantengan una coherencia de diseño a lo largo de todo el conjunto de documentos y permite además su reutilización en futuros proyectos.

Como otra gran ventaja, añadir que permite centralizar el diseño de las páginas, de tal manera que modificar el diseño de todo el Web implicará sólo el cambio de las hojas CSS aplicadas.

Sintaxis de las reglas CSS

Antes de entrar de lleno en la descripción de las reglas de formato CSS definidas en el W3C, pasamos brevemente a describir la sintaxis de los símbolos CSS.

Bloques

Establecen el ámbito donde se definen las marcas de estilo del documento. Según su ubicación serán bloques HTML o bien serán atributos HTML.

```
<STYLE TYPE="text/css">
      <!-- definiciones de formato -->
</STYLE>
<P STYLE="background:#yellow">prueba</P>
```

Código fuente 36

Atributos de estilo CSS

Son pares con el formato **atributo:valor** que nos permitirán indicar qué atributo de estilo queremos modificar y con qué valor. Si hay más de una propiedad que modificar se colocarán separadas por puntos y comas.

La lista de los atributos disponibles en la especificación CSS versión 1 la veremos en el siguiente punto.

```
Color:#000; background: #FFF
```

Código fuente 37

Nota: cuando el estilo se coloca como atributo **style** HTML, debemos encerrar la definición CSS entre comillas.

```
<P STYLE="Color : #000; background : #FFF">prueba</P>
```

Código fuente 38

Estilos CSS

Cuando nos encontramos dentro de un bloque <STYLE>, podremos redefinir las propiedades de los propios elementos estándar de HTML (elementos raíz) y crear nuevas **clases** que podremos emplear para ampliar el formato de los elementos HTML existentes.

Para modificar un elemento raíz, especificaremos el tag HTML y entre corchetes colocaremos los atributos de estilo, como aparece en el Código fuente 39.

```
<STYLE  TYPE="text/css">
BODY
{
  color: #000;
  background: #FFF;
  margin-left: 4%;
```

```
  margin-right: 4%;
  font-family: verdana, helvetica, sans-serif;
}
</STYLE>
```

Código fuente 39

Para crear nuevas **clases** de estilos, dentro del bloque <STYLE> colocaremos el nombre de la clase precedido del símbolo **punto** (".") y después entre corchetes, los modificadores de estilo.

```
<STYLE TYPE="css/text">
.amarillo { background: #ff9 }
.azul   { background: #cff }
.gris   { background: #999 }
</STYLE>
```

Código fuente 40

Nota: Cuando especificamos un elemento raíz, su propia definición ya se aplica al documento, pues permiten modificar el cómo el navegador interpreta los formatos por defecto de cada tag HTML. En el caso de las clases no ocurre así. Es el diseñador del documento el que tiene la responsabilidad de especificar qué tags HTML deben asumir la nueva clase CSS.

```
      <P CLASS="amarillo">Texto con fondo en amarillo</P>
      <P CLASS="azul">Texto con fondo en azul</P>
```

Código fuente 41

Clases y Pseudoclases

Cuando vamos a aplicar estilos a los documentos HTML lo podemos realizar tanto a nivel de **tags HTML** como a nivel de **clases**. Ambos son equivalente con la diferencia de que los tags HTML ya están predefinidos en el lenguaje con su sintaxis y sus estilos por defecto y las **clases** son los nuevos tipos de estilos que vamos a aplicar a las clases HTML ya existentes. Definir estilos es más flexible que actuar *directamente* sobre el tag HTML. Nos pemite aplicarlo cuando es necesario sin preocuparnos de otras hojas de estilos que sobrescriban dicho estilo. Las clases, por tanto, nos posibilitan la reutilización de estilos en varios elementos HTML.

Las subclases son aquellas definiciones de los tags HTML o bien de las clases, que indican un conjunto extendido de funcionalidades del tag HTML / clase. Son referencias a etiquetas de estilo específicas del tag HTML. Veamos el Código fuente 42,por ejemplo.

```
A:link,A:visited,A:active {text-decoration:none}
```

Código fuente 42

La sintaxis de la definición de clases y subclases es:

```
etiqueta:pseudoclase {propiedad1:valor;...;propiedadN:valor}
```

Lo cual nos permitirá aprovechar al máximo para cada elemento el conjunto de "subestilos" que estén programados en el estándar. De echo, este tipo de aproximación es utilizado para implementar los estilos en lo hipervínculos.

Una de las pseudoclases que más juego dan a los estilos es la capacidad que tienen los hipervínculos de saber cuándo se coloca el ratón sobre ellos, momento en el que podemos aprovechar para modificar el estilo y crear efectos gráficos que hacen más dinámico aún este tipo de elementos HTML. Un ejemplo aparece en el Código fuente 43.

```
<STYLE>
  A:link,A:visited,A:active      {color:blue}
  A:hover                        {color:red}
</STYLE>
```

Código fuente 43

Tipos de documentos DOCTYPE

Los estilos han evolucionado mucho en las últimas incorporaciones de las versiones del estándar de HTML. ¿Qué ocurre entonces con las versiones anteriores?. ¿Hay que rescribir el código de estilos de dichas versiones de nuevo?. No es estrictamente necesario. Es posible utilizar el *tag* DOCTYPE para indicar al navegador qué tipo de analizador debe emplear para examinar el código CSS empleado en la página.

Actualmente están vigentes 3 tipos de *validadores* de documentos: la versión *loose* que garantiza compatibilidad hacia atrás con versiones anteriores de la norma, la versión estricta, si queremos que nuestros documentos cumplan siempre la última norma de una manera estricta y la que incluye la nueva especificación XHTML. Para indicar la norma a aplicar en nuestros documentos, colocaremos justo encima de la declaración de <HTML> una de las tres formas descritas:

Versión "loose"

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
       "http://www.w3.org/TR/html4/loose.dtd">
```

Código fuente 44

Versión "Strict"

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
```

Código fuente 45

Versión "XHTML"

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "DTD/xhtml1-strict.dtd">
```

Código fuente 46

Posteriormente, en el bloque <HEAD><HEAD> colocaremos ya la referencia externa a la hoja de estilos correspondiente.

```
<HEAD>
(...)
<LINK REL=STYLESHEET HREF="mi_hoja_estilo.css">
</HEAD>
```

Código fuente 47

Especificación CSS

En base a la documentación del estándar CSS nos encontramos con las declaraciones de estilos siguientes, clasificadas en función de su utilidad.

Propiedades de Fuente y texto	
font-family	font-style
font-variant	font-weight
font-size	@font-face
font	letter-spacing
line-height	text-decoration
text-transform	text-align
text-indent	vertical-align
Propiedades de Color y Fondo	
color	background-color
background-image	background-repeat
background-attachment	background-position

background	
Propiedades de diseño	
margin-top	margin-right
margin-bottom	margin-left
margin	padding-top
padding-right	padding-bottom
padding-left	Padding
border-top-width	border-right-width
Propiedades de diseño (sigue)	
border-bottom-width	border-left-width
border-width	border-top-color
border-right-color	border-bottom-color
border-left-color	border-color
border-top-style	border-right-style
border-bottom-style	border-left-style
border-style	border-top
border-right	border-bottom
border-left	Orderb
float	Clear
Propiedades de Clasificación	
display	list-style-type
list-style-image	list-style-position
list-style	
Propiedades de posicionamiento	
clip	Height

left	Overflow
position	Top
visibility	width
z-index	
Propiedades de impresión	
page-break-before	page-break-after
Propiedades de filtrado	filter
Pseudo-Clases y otras propiedades	
active	hover
@import	!important
cursor	link
visited	
Atributos CSS no soportados	
word-spacing	first-letter pseudo
first-line pseudo	white-space

Tabla 4

Se va a proceder ahora a la descripción de los atributos de estilo que se pueden emplear para dar formato a las páginas HTML con CSS. Como nota, indicar que al explorador de usuario se le suele denominar **UserAgent (UA)** y así lo nombraremos en esta documentación.

Background	
Valores	<background-color>, <background-image>, <background-repeat>, <background-attachment>, <background-position>
Inicial	Sin valor inicial
Heredado	No
Establece la propiedad del papel de fondo empleado en las páginas HTML. Sirve como abreviatura para acceder a las propiedades del tapiz desde el mismo lugar. Ejemplo: `P { background: url(chess.png) gray 50% repeat fixed }`	
Background - attachment	
Valores	Scroll,fixed

Inicial	Scroll
Heredado	No

Indica que si hay una imagen de fondo se empleará el Canvas para dibujar la imagen anclada en el fondo o bien debe hacer scroll junto con el documento

Ejemplo:

```
BODY

{
    background: red url(pendant.gif);
    background-repeat: repeat-y;
    background-attachment: fixed;
}
```

Background-color

Valores	<color>, transparent
Inicial	Transparent
Heredado	No

Establece el color de fondo de un elemento HTML

Ejemplo: H1 { background-color: #F00 }

Background-image

Valores	<URL>,none
Inicial	None
Heredado	No

Establece imagen de fondo de un elemento HTML. Se aconseja especificar un color de fondo por defecto en el caso de que la imagen no pueda ser cargada. Si la imagen si es cargada, se colocará sobrepuesta al color de fondo.

Ejemplo: BODY { background-image: url(marble.gif) }

P { background-image: none }

Background-position

Valores	<percen*tage*>, <length>, top, center, bottom, left, center, righ
Inicial	0% 0%
Heredado	No

Si se ha especificado una imagen de fondo, esta propiedad establece la posición inicial del tapiz.

Con el valor 0% 0% se coloca en la coordenada (0,0).El par 100% 100% colocaría el origen en la esquina abajo-derecha –depende de la resolución del monitor-.

Con los valores en cm se colocan a (x,y) centrímetros del borde del User Agent (navegador). Ejemplo el par 2cm 2cm

Existen combinaciones predefinidas en los valores top,left,right,center,bottom

'top left' y 'left top' equivalen a '0% 0%'.

'top', 'top center' y 'center top' equivalen a '50% 0%'.

'right top' y 'top right' equivalen a '100% 0%'.

'left', 'left center' y 'center left' equivalen a '0% 50%'.

'center' y 'center center' equivale a '50% 50%'.

'right', 'right center' y 'center right' equivale a '100% 50%'.

'bottom left' y 'left bottom' equivale a '0% 100%'.

'bottom', 'bottom center' y 'center bottom' equivale a '50% 100%'.

'bottom right' y 'right bottom' equivale a '100% 100%'.

Ejemplo:

```
BODY { background: url(banner.jpeg) right top }    /* 100%   0% */

BODY { background: url(banner.jpeg) top center }   /*  50%   0% */

BODY { background: url(banner.jpeg) center }       /*  50%  50% */

BODY { background: url(banner.jpeg) bottom }       /*  50% 100% */
```

Background-repeat	
Valores	repeat, repeat-x, repeat-y, no-repeat
Inicial	Repeat
Heredado	No

Si se ha especificado una imagen de fondo, establece cuantas veces se repite dicha imagen. Si se especifica:
“repeat”: la imagen se repite tanto vertical como horizontalmente
“repeat-x”, “repeat-y”: se repite horizantal (eje X) o verticalmente (eje Y)
“no repeat”: La imagen no se repite

Ejemplo:

```
BODY {
    background: red url(pendant.gif);
    background-repeat: repeat-y;
    }
```

Border	
Valores	<border-width>, <border-style>, <color>
Inicial	No definido
Heredado	No
Esta propiedad es un atajo para establecer las propiedades por defecto de todos los elementos HTML que emplean border para su apariencia gráfica. Permite establecer el ancho del borde (*border-width*), el estilo del borde (*border-style*) y su color (*color*) Ejemplo: P {border: solid red} NOTA: si queremos detallar un tipo de borde, usaremos los estilos específicos de cada borde. Los cuales vemos justo a continuación. El orden es importante. El "overlapping" afecta a aquellos elementos que no estén completamente detallados, asumiendo las propiedades por defecto establecidas en algún lugar de la definición.	
Border-bottom	
Valores	<border-width>, <border-style>, <color>
Inicial	No definido
Heredado	No
Esta propiedad establece el ancho, estilo y color de un borde inferior Ejemplo: P {border-bottom: solid red}	
Border-bottom-width	
Valores	thin, medium, thick, <length>
Inicial	medium
Heredado	No
Esta propiedad establece el ancho de un borde inferior Ejemplo: P {border-bottom-width: thick}	
Border-color	
Valores	<color>
Inicial	Valor de la propiedad color
Heredado	No
Esta propiedad establece el color de los cuatro bordes. Admite desde un valor hasta cuatro –uno para cada borde- Ejemplo: P { color: black; background: white; border: solid; }	
Border-left	
Valores	<border-width>, <border-style>, <color>
Inicial	No definido
Heredado	No
Esta propiedad establece el ancho, estilo y color de un borde izquierdo Ejemplo: P {border-left: solid red}	
Border-left-width	
Valores	thin, medium, thick, <length>
Inicial	medium
Heredado	No

Esta propiedad establece el ancho de un borde izquierdo Ejemplo: P {border-left-width: thick}	
Border-right	
Valores	<border-width>, <border-style>, <color>
Inicial	No definido
Heredado	No
Esta propiedad establece el ancho, estilo y color de un borde derecho Ejemplo: P {border-right: solid red}	
Border-right-width	
Valores	thin, medium, thick, <length>
Inicial	medium
Heredado	No
Esta propiedad establece el ancho de un borde derecho Ejemplo: P {border-right-width: thick}	
Border-style	
Valores	none, dotted, dashed, solid, double, groove, ridge, inset, outset
Inicial	Ninguno
Heredado	No
Esta propiedad establece el estilo de los cuatro bordes de un elemento HTMl. Puede contiener desde uno hasta los cuatro valores. En el caso de ir los cuatro valores, seguirían el mismo orden que el comentado para el atributo "*border-width*". Los estilos de borde significan: **none** ningún tipo de borde es dibujado **dotted** linea de puntos suspensivos **dashed** linea de rayas discontinuas **solid** Linea solida **double** Linea doble. La suma de las dos lineas y su espacio es el establecido en <border-width> **groove** efecto de hundido 3D basado en el color establecido en <color> **ridge** efecto de relieve 3D basado en el color establecido en <color> **inset** similar a groove **outset** similar a ridge Ejemplo: #xy34 { border-style: solid dotted }	
Border-top	
Valores	<border-top-width>, <border-style>, <color>
Inicial	Ninguno
Heredado	No
Esta propiedad es un "acceso rápido" a las propiedades del borde superior de un elemento HTML. Permite establecer el ancho, estilo y color del borde superior Notas: las propiedades no establecidas asumiran los valores por defecto. Si <color> no es establecido se asumirá el <color> por defecto Ejemplo: H1 {border-top: thick solid red}	
Border-top-width	
Valores	thin, medium, thick, <length>
Inicial	medium
Heredado	No

Esta propiedad establece el ancho de un borde superior

Ejemplo:
P {border-top-width: thick}

Border-width	
Valores	thin, medium, thick, <length>
Inicial	Ninguno
Heredado	No

Esta propiedad actúa como acceso rápido a las propiedades <border-width-top>,<border-width-left>,<border-width-right>,<border-width-bottom>. Puede tener desde uno hasta los cuatro valores para cada uno de los bordes.

Nota: Si se especifica SOLO un valor, se asumirá para los cuatro bordes. Si se especifican dos, el segundo se asumirá para los dos restantes... y así sucesivamente.

Ejemplo:

```
H1 { border-width: thin }                /* thin thin thin thin */

H1 { border-width: thin thick }          /* thin thick thin thick */

H1 { border-width: thin thick medium }   /* thin thick medium thin */

H1 { border-width: thin thick medium thin } /* thin thick medium thin */
```

Clear	
Valores	none, left, right, both
Inicial	None
Heredado	No

Esta propiedad establece como debe posicionarse un elemento 'flotante' con respecto a elementos de alrededor. Concretamente, establece la posición en la cual no se permite la ubicación de un elemento.
Si se establece a 'left', los elementos que se encuentren a la izquierda se moverán a una línea de colocación que no se encuentre a la izquierda. Si se especifica 'none', se podrán posicionar los elementos en cualquier lugar.

Ejemplo:
H1 { clear: left }

Color	
Valores	Un color
Inicial	Especifico del Explorador (o User Agent)
Heredado	No

Esta propiedad establece un color de primer plano al elemento. Se puede especificar de dos maneras: como una cadena ASCII con el nombre estandarizado del color, o bien basado en un valor RGB.

Ejemplo:

```
H1 { color: red }            /* lenguaje natural */
H1 { color: rgb(255,0,0) }   /* RGB rango 0-255  */
```

Display	
Valores	block, inline, list-item, none
Inicial	Block
Heredado	No

Esta propiedad describe el cómo/ y si se dibuja un elemento en el **canvas** de la página.

Block Abre una nueva caja, posicionada en coordenadas relativas a CSS. Los elementos <H1> y <P> suelen estar marcados como tipo **block**.
Inline amplía el ámbito de una caja definida ajustándose a los tipos de dicha caja.
List-item Equivalente **block** excepto en que se añade un marcador de elemento. El elemento <LI> es el que lo suele implementar.
None oculta el dibujo del elemento, sus hijos y su linea de cuadro de alrededor.

Notas: El UA puede ignorar 'display' y utilizar las especificaciones estándar de cada elemento

Ejemplo:

```
P { display: block }
EM { display: inline }
LI { display: list-item }
IMG { display: none }   /* invalidar todas las imagenes del documento */
```

Float

Valores	left, right, none
Inicial	Ninguno
Heredado	No

Con el valor de 'none'el elemento aparecerá tal y como está definido.

Si 'left'/'right' están especificados, el texto se alineará hacia la izquierda / derecha colocándose a la derecha / izquierda (respectivamente) de un elemento

Ejemplo:

```
IMG.icon {
    float: left;
    margin-left: 0;
    }
```

Font

Valores	<font-style>, <font-variant>, <font-weight> <font-size> <line-height> <font-family>
Inicial	Ninguno
Heredado	Si

Especifica el estilo gráfico de una fuente de letras.

Ejemplo:

```
P { font: 12pt/14pt sans-serif }
P { font: 80% sans-serif }
P { font: x-large/110% "new century schoolbook", serif }
P { font: bold italic large Palatino, serif }
P { font: normal small-caps 120%/120% fantasy }
```

Font-family

Valores	<family-name>, <generic-family>
Inicial	Ninguno
Heredado	Si

Establece el conjunto de tipos de fuentes disponibles . Se pueden establecer en base a :
<family-name>
Fuente específica como por ejemplo "gill" y "helvetica"
<generic-family>

Se definen como familias genéricas en el estandar:
'serif' (por ejemplo, Times)
'sans-serif' (por ejemplo, Helvetica)
'cursive' (por ejemplo, Zapf-Chancery)
'fantasy' (por ejemplo,. Western)
'monospace' (por ejemplo, Courier)

Ejemplo:
BODY { font-family: "new century schoolbook", serif }
<BODY STYLE="font-family: 'My own font', fantasy">

Font-size	
Valores	<absolute-size>, <relative-size>, <length>, <percen*tag*e>
Inicial	Médium
Heredado	Si

<absolute-size> indica una posición de un tamaño de fuente almacenado en tablas precalculadas en el navegador para representar las fuentes. Se puede indicar en puntos o bien empleando las macros [xx-small | x-small | small | medium | large | x-large | xx-large]. Dependerá de la resolución del sistema el que se disponga de más o menos calidad.

<**relative-size>** Posición relativa en la tabla de tamaños de fuentes en función al tamaño del elemento Padre. Puede coger los valores: [large | small]. El navegador extrapolará el tamaño en función de la resolución y el tamaño del elemento Padre.

<**length>** y **<percen*tag*e>** no tienen en consideración las tablas para presentar el tamaño de la fuente.

Se pueden emplear tambien las unidades de medida “em” y “ex” para incrementar o decrementar en unidades la fuente actual respecto a la fuente del padre.

Ejemplos:
P { font-size: 12pt; }
BLOCKQUOTE { font-size: larger }
EM { font-size: 150% }
EM { font-size: 1.5em }

Font-style	
Valores	normal, italic, oblique
Inicial	Normal
Heredado	Si

Selecciona, dentro de una familia de fuentes, el estilo gráfico que se emplea para representarla (negrita, cursiva, itálica o aplicar un efecto especial calculado-no pregenerado).

Ejemplos:
H1, H2, H3 { font-style: italic }
H1 EM { font-style: normal }

Font-variant	
Valores	normal, small-caps
Inicial	Normal
Heredado	Si

Tipo de variación de la fuente que permite representar un tamaño más pequeño del que se representa en una fuente normal. Permite generar efectos de palabras en minúsculas que con su tamaño resaltan determinadas características.

Ejemplos:
H3 { font-variant: small-caps }
EM { font-style: oblique }

Font-weight	
Valores	normal, bold, bolder, lighter, 100, 200, 300, 400, 500, 600, 700, 800, 900
Inicial	Normal
Heredado	Si

Establece el grosor de la fuente de tal manera que cada medida es tan oscura o más que su antecesora. "normal" equivaldría al valor 100 en lo que "bold" equivale a un valor de 700. "bolder" y "lighter" permiten establecer el ancho en función de los valores de la fuente padre.

No todas las familias de fuentes soportan todas las medidas de ancho especificados en la norma. Para más información, conviene consultar la tabla de anchos por fuente.
Si no se dispone de un equivalente para una fuente determinada, el explorador interpolará al tamaño más próximo.

Ejemplos:
P { font-weight: normal } /* 400 */
H1 { font-weight: 700 } /* bold */
STRONG { font-weight: bolder } /* ejemplo de aprovechamiento de la herencia */

Height	
Valores	<length>, auto
Inicial	Auto
Heredado	No

Esta propiedad se puede aplicar a las fuentes, pero su uso más común es poder establecer las propiedades de altura de la imágenes del documento. Valores negativos no están permitidos. "auto" asume las propiedades de la propia imagen.

Ejemplo:
IMG.icon { height: 100px }

Letter-spacing	
Valores	normal, <length>
Inicial	Normal
Heredado	Si

Permite establecer el espaciado entre caracteres. Se pueden especificar valores negativos, pero es muy específico de la plataforma. El Explorador es libre de elegir el algoritmo de espaciado y puede estar influenciado por el tipo de alineación elegida.

Ejemplo:
BLOCKQUOTE { letter-spacing: 0.1em }
Nota: a veces los exploradores intentan justificar el texto cuando se establece el atributo como "normal". Para evitarlo, será necesario establecer un valor absoluto, antes que "normal":
BLOCKQUOTE { letter-spacing: 0 }
BLOCKQUOTE { letter-spacing: 0cm }

Line-height

Valores	normal, <number>, <length>, <percen*tage*>
Inicial	Normal
Heredado	Si

Permite establecer el espaciado entre líneas. Cuando se especifica un valor absoluto, la separación se establece en base al tamaño de la fuente multiplicado por el factor de la anchura de la línea base.

Los valores negativos no están permitidos.

Ejemplo:

```
/* tres líneas con el mismo resultado final … */
    DIV { line-height: 1.2; font-size: 10pt }     /* number */
    DIV { line-height: 1.2em; font-size: 10pt }   /* length */
    DIV { line-height: 120%; font-size: 10pt }    /* percentage */
```

Line-style-image

Valores	<url>, none
Inicial	None
Heredado	Si

Establece la imagen que se va a utilizar como marcador de los elementos de una lista.

Ejemplo:

```
UL { list-style-image: url(http://png.com/ellipse.png) }
```

Line-style-position

Valores	inside, outside
Inicial	Outside
Heredado	Si

Determina como se va a colocar el marcador de elementos con respecto a los elementos de la lista. “inside” lo coloca con un margen respecto al título de la lista y “outside” lo coloca sin margen con respecto al título de la lista

Ejemplo:

```
li.none {list-style-position: inside}
```

Line-style-type

Valores	disc, circle, square, decimal, lower-roman, upper-roman, lower-alpha, upper-alpha, none
Inicial	Disc
Heredado	Si

Determina el estilo de representación que se va a utilizar como elemento diferenciador de las listas.

Será aplicable siempre y cuando no se especifique una imagen como separador o bien la URL de la imagen no esté disponible.

Disc,circle y square : especifican un carácter de “viñeta”
Lower-roman, upper-roman : especifican numeración romana p.Ej: ix, IX
Lower-alpha, upper-alpha : especifican alfabeto p.Ej: a , A
None : no se emplea marcador de separación

Ejemplo:

```
OL { list-style-type: decimal }       /* 1 2 3 4 5 etc. */
OL { list-style-type: lower-alpha }   /* a b c d e etc. */
OL { list-style-type: lower-roman }   /* i ii iii iv v etc. */
```

Margin

Valores	<length>, <percen*tage*>, auto
Inicial	0

Heredado	No

Es el acceso rápido a todas las propiedades establecidas por "margin-top", "margin-right", "margin-bottom" y "margin-left".

Si se especifican los cuatro valores, afecta a los cuatro márgenes. Si sólo se especifica un valor, éste afectará a todos los márgenes.

Nota: si alguna no se especifica, se asumirá como valor por defecto, el establecido en su valor opuesto (ejemplo: si asigno el margin-left y no el "right", éste coje el valor de "left"). Admite valores negativos, pero son específicos del navegador el cómo se representen.

Ejemplo:
BODY { margin: 2em } /* all margins set to 2em */
BODY { margin: 1em 2em } /* top & bottom = 1em, right & left = 2em */
BODY { margin: 1em 2em 3em } /* top=1em, right=2em, bottom=3em, left=2em */

Margin-bottom	
Valores	<length>, <percen*tag*e>, auto
Inicial	Ninguna. Se establece con propiedades por defecto
Heredado	No

Establece el margen inferior de un elemento

Ejemplo:
H1 { margin-bottom: 3px }

Margin-left	
Valores	<length>, <percen*tag*e>, auto
Inicial	Ninguna. Se establece con propiedades por defecto
Heredado	No

Establece el margen izquieredo de un elemento

Ejemplo:
H1 { margin-left: 2em }

Margin-rigth	
Valores	<length>, <percen*tag*e>, auto
Inicial	Ninguna. Se establece con propiedades por defecto
Heredado	No

Establece el margen derecho de un elemento

Ejemplo:
H1 { margin-right: 12.3% }

Margin-top	
Valores	<length>, <percen*tag*e>, auto
Inicial	Ninguna. Se establece con propiedades por defecto
Heredado	No

Establece el margen superior de un elemento

Ejemplo:
H1 { margin-top: 2em }

Padding

Valores	<length>, <percen*tag*e>
Inicial	Ninguna. Se establece con propiedades por defecto
Heredado	No

Acceso rápido a las propiedades “padding-top”, “padding-right”, “padding-bottom”, “padding-left” desde el mismo atributo. Si se aplican los cuatro valores se establecen los cuatro valores; si se especifica un valor, se toma como por defecto para los cuatro. Si algún valor falta, se asume como por defecto su opuesto.
Establece el espacio de separación con respecto a los bordes del elemento.

Notas: Los valores no pueden ser negativos

Ejemplo:

```
H1 {
    background: white;
    padding: 1em 2em;
  }
```

Padding-bottom	
Valores	<length>, <percentage>
Inicial	0
Heredado	No

Establece el espacio de separación con respecto al borde inferior de un elemento.

Ejemplo:
BLOCKQUOTE { padding-bottom: 2em }

Padding-left	
Valores	<length>, <percentage>
Inicial	0
Heredado	No

Establece el espacio de separación con respecto al borde izquierdo de un elemento.

Ejemplo:
BLOCKQUOTE { padding-left: 20% }

Padding-rigth	
Valores	<length>, <percentage>
Inicial	0
Heredado	No

Establece el espacio de separación con respecto al borde derecho de un elemento.

Ejemplo:
BLOCKQUOTE { padding-bottom: 10px }

Padding-top	
Valores	<length>, <percentage>
Inicial	0
Heredado	No

Establece el espacio de separación con respecto al borde superior de un elemento.

Ejemplo:
BLOCKQUOTE { padding-top: 0.3em }

Text-align	
Valores	left, right, center, justify
Inicial:	Especifico del navegador
Heredado	Si

Establece el cómo se alinea un elemento HTML.

El valor por defecto depende del Agente del Usuario (navegador) y del tipo de lenguaje humano del sistema operativo (p.Ejemplo: el Español se lee de izquierda a derecha, luego por defecto alineado a la izquierda).

Ejemplo:
DIV.center { text-align: center }

En este ejemplo se asume que todos los elementos contenidos dentro de un *tag* "DIV" aparecerán centrados en su cuerpo.

Text-decoration	
Valores:	None, underline, overline, line-through, blink
Inicial:	None
Heredado:	No *

Describe el tipo de decoración gráfica que se usará para mostrar el texto del usuario. Si el elemento no tiene texto (por ejemplo un *tag* IMG) o bien es un *tag* vacío (<EM></EM>), esta propiedad no tendrá efecto.
Los colores del efecto de decoración se tomarán del estilo "color" definido en el documento.

La propiedad no es hereditaria, pero puede ocurrir que algunos elementos puedan coincidir con su padre. Por ejemplo, si un elemento tiene efecto de subrayado, éste permanecerá también sobre los elementos contenidos. Y con el mismo color del contenedor, a pesar de que los hijos tengan un color distinto.

Underline : efecto de subrayado
Overline : efecto de subrayado superior
Line-through : efecto de tachadura
Blink : efecto de parpadeo. Su efecto depende del navegador.

Ejemplo:
A:link, A:visited, A:active { text-decoration: underline }

En este ejemplo se ve el efecto comentado de la herencia. Los *tag* <A> afectan también a todos los elementos que contienen HREF. (que ya tienen subrayado por defecto por el mero echo de ser hipervínculos). Es un caso muy concreto.

Text-indent	
Valores	<length>, <percen*tag*e>
Inicial	0
Heredado	Si

Especifica el valor de indentación que se va a aplicar sobre el primer elemento de un párrafo. Este valor sólo afectará al primer elemento y no a aquellos que se encuentren tras un separador como <BR>.

Ejemplo:
P { text-indent: 3em }

Text-transform	
Valores	capitalize, uppercase, lowercase, none
Inicial	Ninguno
Heredado	Si

Capitalize : pone en mayúscula el primer caracter de cada palabra
Uppercase : pone en mayúculas todos los caracteres de una palabra
Lowercase : pone en minúsculas todos los caracteres de una palabra
None : neutraliza cualquier posible valor anterior o por defecto.

Ejemplo:
H1 { text-transform: uppercase }
Nota: el Agente de usuario puede ignorar esta propiedad si el juego de caracteres empleado no es compatible con el Latin-1. Para otros juegos de caracteres, se escogerá el algoritmo de transformación basado en el ASCII Unicode.

White-space

Valores	normal, pre, nowrap
Inicial	Normal
Heredado	Si

Establece como se trata el carácter del espacio cuando se produce una rotura en línea de una frase al llegarse al límite derecho de escritura.
"normal" : deja el carácter en blanco tal cual se escribe en la frase y salta a la siguiente línea la palabra siguiente.
"pre" : elimina el espacio en blanco al final de la frase y genera el salto de línea
"nowrap" : no se produce rotura de frase. Se amplía la región de escritura

Ejemplo:
PRE { white-space: pre }
P { white-space: normal }
Nota: el agente de usuario puede ignorar esta propiedad e implementar los mecanismos establecidos como por defecto en la norma HTML.

Width

Valores:	<length>, <percen*tag*e>, auto
Inicial:	Auto
Heredado:	No

Esta propiedad puede ser aplicada a texto, pero su utilidad real es establecer el ancho de imágenes y tablas / celdas de tablas. En el caso de las imágenes se fuerza a realizar un escalado de la misma. Como ampliación, indicar que si "height" se establece a "auto", se preservará el factor de escalado para que la altura de la imagen se adapte al nuevo ancho.
Valores negativos no son admitidos.
Ejemplo:
IMG.icon { width: 100px }
Notas: si tanto "height" como "width" se dejan en "auto", se asumirán como valores los que tenga definidos la propia imagen. Valor por defecto de dichas propiedades.

Word-spacing

Valores:	normal, <length>
Inicial:	Normal
Heredado:	Si

Establece unidades de medida de separación adicional entre las palabras de un texto. Los valores pueden ser negativos, pero son específicos de la plataforma. El Agente de usuario es libre de seleccionar el algoritmo de separación en función al tipo de fuente y decoración, así como por alineación del texto (por ejemplo: efectos de justificación).
Ejemplo:
H1 { word-spacing: 1em }
Notas: el Agente de usuario puede interpretar cualquier valor especificado como atributo "normal".

Curiosidades y nuevos estilos…

Para terminar este capítulo, sólo mostrar un estilo adicional, que es compatible con el navegador Microsoft Internet Explorer en su versión 5.0 o superior, que ya permite incluso interactuar con los elementos activos del propio navegador: el elemento SCROLLBAR. Y qué mejor que documentarlo con el ejemplo que aparece en la Tabla 5.

```
<STYLE>
BODY
{
    SCROLLBAR-FACE-COLOR: #5f5f5f;
    SCROLLBAR-HIGHLIGHT-COLOR: #606060;
    SCROLLBAR-SHADOW-COLOR: #000000;
    SCROLLBAR-3DLIGHT-COLOR: #7d7d7d;
    SCROLLBAR-ARROW-COLOR: #ff0909;
    SCROLLBAR-TRACK-COLOR: #000000;
    SCROLLBAR-DARKSHADOW-COLOR: #000000
}

</STYLE>
```

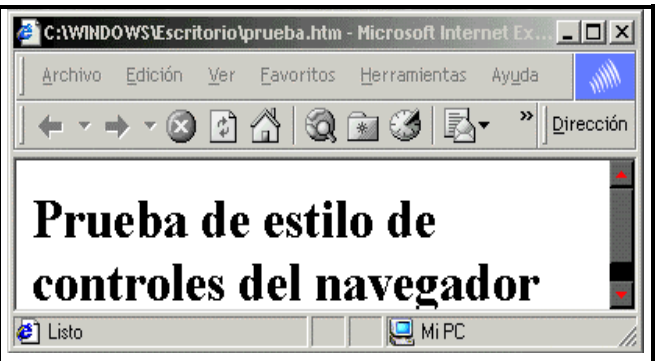

Tabla 5

Caso Práctico

Definición de la práctica

Vamos a proceder a diseñar una página WEB en función de las modas impuestas actualmente. En dicha "moda" premia por encima de todo el mostrar la mayor cantidad de información posible. Por ello los elementos tradicionales de HTML se quedan un poco "cortos" a no ser que contemos con los estilos CSS.

Montaremos la página de inicio de "El actual", un nuevo tipo de publicación virtual que quiere poner al día al intrépido cibernauta. De tal manera que el diseño quedará como se ve en la figura.

Como premisas:

- los hipervínculos deben activarse al ponerse el ratón encima y no estarán subrayados
- Las tablas de datos tendrán bordes externos y su texto tiene que estar justificado
- Las tablas tendrán un encabezado centrado y de color amarillo de fondo y la fuente en negrita y cursiva
- Todos los textos que se encuentran cerca de una imagen tendrán que estar alineados al centro de la altura de dicha imagen
- La fuente del documento estará basada en la familia de "Verdana"
- Se dejarán márgenes del 2% respecto a los bordes del documento
- El fondo del documento llevará un fondo corporativo que además se desea que esté fijo en la pantalla y no se mueva al hacer Scroll del documento (en el caso de que esto sea posible).
- Los botones corporativos serán de color azul
- Al final irá información de copyright en cursiva y con un tamaño de 7pt

La forma final de la página...

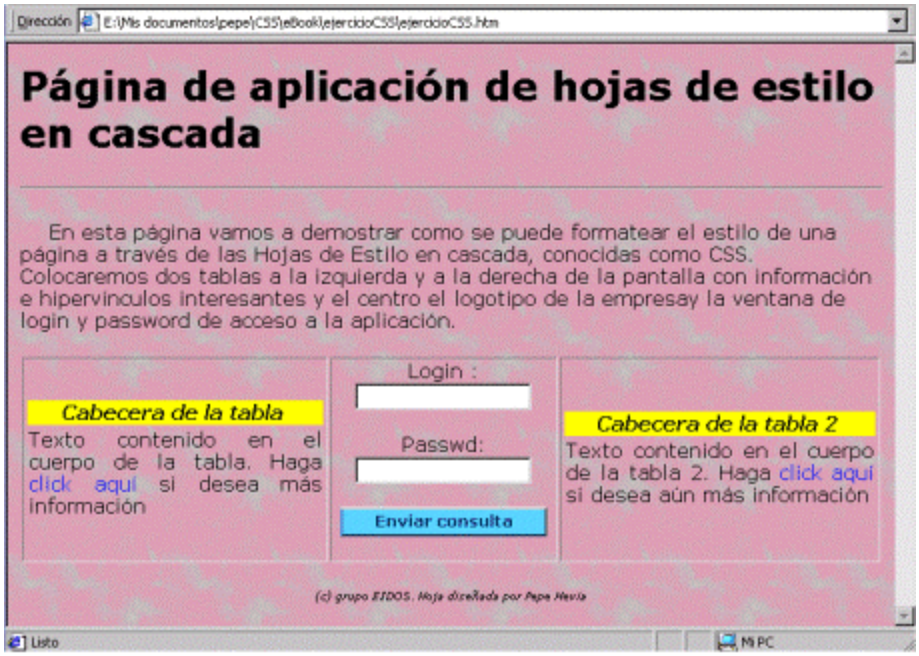

Figura 26. La página en su fase final.

El diseño inicial de la página...

Como se comentó en el tema de CSS, el primer paso es diseñar el documento HTML tal y como debe representar la información, pero sin **ningun** tipo de formato gráfico. Puesto que nuestro equipo ya tiene los estilos oportunos creados y documentados. Por lo tanto, la hoja en su primera fase de diseño quedará como muestra la Figura 27

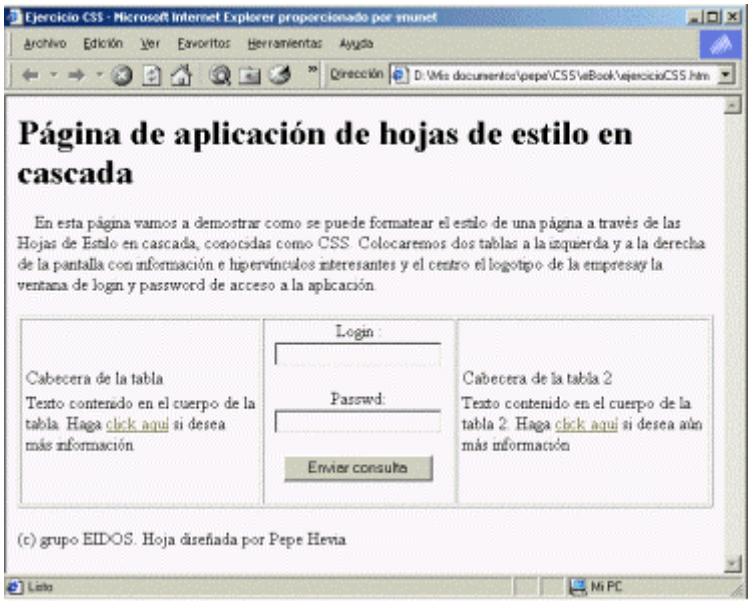

Figura 27. La página en su fase inicial.

Como podemos observar, el diseño es únicamente un conjunto de tags HTML que proporcionan la forma del documento. Pero hasta aquí no nos preocupamos de cómo debe mostrarse. Solo de qué debe mostrarse. Esta aproximación nos permitirá ganar en flexibilidad en futuros cambios de la página.

El código de la página, queda reflejado en el fuente HTML que muestra el Código fuente 48.

```
<html>
<head>
<title>Ejercicio CSS</title>
</head>
<body>
<h1>Página de aplicación de hojas de estilo en cascada</h1>
<HR>
<p>    En esta página vamos a demostrar como se puede formatear
el estilo de una página a través de las Hojas de Estilo en cascada, conocidas
como CSS. Colocaremos dos tablas a la izquierda y a la derecha de la pantalla
con información e hipervínculos interesantes y el centro el logotipo de la
empresay la ventana de login y password de acceso a la aplicación.</p>
<Table border=1>
<TR>
       <TD>
              <TABLE>
                <TR>
                     <TD>
                            Cabecera de la tabla
                     </TD>
                </TR>
                <TR>
                     <TD>
                            Texto contenido en el cuerpo de la tabla.
                            Haga <A HREF="#">click aquí</A> si desea más información
                     </TD>
                </TR>
          </TABLE>
       </TD>
       <TD>
              <FORM>
                 <CENTER>
                     Login : <INPUT TYPE="TEXT" NAME="LOGIN">
             </CENTER>
                     <BR>
                     <CENTER>
                     Passwd: <INPUT TYPE="PASSWORD" NAME="LOGIN">
                     </CENTER>
                     <BR>
                     <center>
                            <INPUT TYPE="SUBMIT" NAME="ENVIAR">
                     </center>
              </FORM>
       </TD>
       <TD>
              <TABLE>
                <TR>
                     <TD>
                     Cabecera de la tabla 2
                     </TD>
                </TR>
                <TR>
                <TD>
                            Texto contenido en el cuerpo de la tabla 2.
                            Haga <A HREF="#">click aquí</A> si desea aún más
información
                     </TD>
                </TR>
```

```
          </TABLE>
       </TD>
</TR>
</TABLE>
<p>(c) grupo EIDOS. Hoja diseñada por Pepe Hevia</p>
</body>
</html>
```

Código fuente 48. Código HTML de la página en su fase inicial

Aplicando estilos. Análisis de requisitos.

De los requisitos especificados en el enunciado, podemos deducir que nos encontremos dos maneras de definir los estilos:

a) los que afectan no a esta página sino a todas: son aquellos que establecen los estilos "corporativos". Por tanto, es necesario que los podamos reutilizar. Con lo cual los definiremos en ficheros independientes a la página, con extensión **.CSS**. Y además, intentaremos agrupar los estilos por objetivos y funcionalidades:

Afectan al documento en sí DOCUMENTO.CSS

- Todos los textos que se encuentran cerca de una imagen tendrán que estar alineados al centro de la altura de dicha imagen
- La fuente del documento estará basada en la familia de "Verdana"
- Se dejarán márgenes del 2% respecto a los bordes del documento
- El fondo del documento llevará un fondo corporativo que además se desea que esté fijo en la pantalla y no se mueva al hacer Scroll del documento (en el caso de que esto sea posible).

Afectan a hipervínculos VINCULOS.CSS

- los hipervínculos deben activarse al ponerse el ratón encima y no estarán subrayados

Afectan a tablas TABLAS.CSS

- Las tablas de datos tendrán bordes externos y su texto tiene que estar justificado
- Las tablas tendrán un encabezado centrado y de color amarillo de fondo y la fuente en negrita y cursiva

Afectan a controles CONTROLES.CSS

- Los botones corporativos serán de color azul

b) Y los que son específicos de la página, que irán o bien como bloque o bien como IN-LINE. En nuestro caso, que es muy específico, será IN-LINE.

- Al final de la página principal irá información de copyright que irá cursiva y con un tamaño de 7pt y tipo de letra Arial.

Pues manos a la obra....

Afectan al documento en sí DOCUMENTO.CSS

- Todos los textos que se encuentran cerca de una imagen tendrán que estar alineados al centro de la altura de dicha imagen

```
IMG
{   text-align=center;
}
```

Código fuente 49

- La fuente del documento estará basada en la familia de "Verdana"
- Se dejarán márgenes del 2% respecto a los bordes del documento
- El fondo del documento llevará un fondo corporativo que además se desea que esté fijo en la pantalla y no se mueva al hacer Scroll del documento (en el caso de que esto sea posible) y que esté centrado en el fondo del escritorio.

```
BODY
{   font-family:Verdana;
    background-image:url(fondo.gif);
    background-position:center;
    background-attachment:fixed;
    margin:2%;
}
```

Código fuente 50

Afectan a hipervínculos VINCULOS.CSS

- los hipervínculos deben activarse al ponerse el ratón encima, y no deben tener el subrayado

```
A,A:link,A:vlink,A:active
{   color:blue;
    font-family:Verdana;
    text-decoration:none;
}
A:hover
{   color:red;
    font-family:Verdana;
    text-decoration:none;
}
```

Código fuente 51

Afectan a tablas TABLAS.CSS

- Las tablas de datos tendrán bordes externos y su texto tiene que estar justificado

```
TABLE
{   border:1;
    text-align:justify;
```

```
}
```

Código fuente 52

- Las tablas tendrán un encabezado centrado y de color amarillo de fondo y la fuente en negrita y cursiva. En este caso, se define como clase ya que no podemos depender de si el usuario utiliza el tag **<TH>** para definir las cabeceras.

```
.CABECERA
{   text-align:center;
    background-color:yellow;
    font-weight:bold;
    font-style:italic;
}
```

Código fuente 53

Con lo que habrá que modificar en el código, las líneas que muestra el Código fuente 54.

```
<TD CLASS="CABECERA"> Cabecera de la tabla   </TD>   (...)
<TD CLASS="CABECERA"> Cabecera de la tabla 2 </TD>   (...)
```

Código fuente 54

Afectan a controles CONTROLES.CSS

- Los botones corporativos serán de color azul.Como podremos encontrarnos con <INPUT STYLE="Button"> o bien el tag <BUTTON>, para hacerlo más flexible, como clase.

```
.BOTON_CORP
{   BORDER-RIGHT: #ffffff 2px outset;
    BORDER-TOP: #ffffff 2px outset;
    FONT-WEIGHT: bold;
    BORDER-LEFT: #5895ff 2px outset;
    CURSOR: hand;
    COLOR: #003366;
    BORDER-BOTTOM: #5895ff 2px outset;
    FONT-FAMILY: "Verdana";
    BACKGROUND-COLOR: #58d2ff
}
```

Código fuente 55

Con lo que habrá que modificar la línea de la página original por la que muestra el Código fuente 56.

```
<INPUT TYPE="SUBMIT" CLASS="BOTON_CORP">   (...)
```

Código fuente 56

Por último, nos tocará definir el estilo del copyright. Como se especifica en el enunciado, ”Al final de la página principal irá información de copyright que irá cursiva y con un tamaño de 7pt y tipo de letra Arial”. Con lo cual, si echamos un ojo al código original, veremos que va dentro de un párrafo. Por tanto, asignaremos al tag <P> del copyright, los datos de estilo que muestra el Código fuente 57.

```
<P ALIGN=center STYLE=”font-style:italic;font-size:7pt;font-family:Verdana”>
  (c) grupo EIDOS. Hoja diseñada por Pepe Hevia
</p>
```

Código fuente 57

Quedando finalmente la página como ya se mostró en la Figura 26. Por último, especificar como quedará la estructura del directorio:

C:\ EJERCICIOCSS --- \ejercicio.htm

--- \documento.css

--- \vinculos.css

--- \tablas.css

--- \controles.css

--- \fondo.gif

3

Introducción a Javascript

Introducción

Ya hemos hablado en el primer capítulo del papel de los lenguajes de script en las páginas web de cliente: capacitan la ejecución de llamadas a los objetos creados por DOM, y permiten, por tanto, la creación de páginas ricas en contenido dinámico, que son capaces de responder a las necesidades actuales de programación, aportando recursos multimedia, estilos dinámicos, capacidad de cálculo, etc.

También hemos comentado que, dentro de los lenguajes de *script* más populares, *Javascript* ocupa un puesto destacado debido a su portabilidad y a su soporte por parte de los navegadores más populares del mercado: Internet Explorer, Netscape Navigator y Opera. Esto, unido al hecho de que Javascript sea también un estándar, hacen de este lenguaje el candidato idóneo para la construcción de páginas dinámicas de cliente.

Oficialidad, Autoría y Estandarización

En cuanto al aspecto oficial del establecimiento de estándares, hay que recordar que –en este caso- no es la W3C la encargada de su promoción y publicación como estándar, sino otra entidad de normalización: ECMA. El lector interesado, puede consultar la página web oficial del sitio ECMA (www.ecma.ch) y descargar la especificación, tal y como está publicada allí, al ser un documento público. Respecto a su autoría, el propio documento oficial del estándar nos dice que:

> *"Este estándar ECMA está basado en varias tecnologías originales bien conocidas, entre las que destacan JavaScript (Netscape), y JScript (Microsoft). El lenguaje fue inventado*

*por **Brendan Eich**, de Netscape y apareció por primera vez en la versión 2.0 de Navigator. Ha continuado apareciendo en todos los navegadores posteriores, así como en todos los navegadores de Microsoft, comenzando por la versión 3.0 de Internet Explorer.*

El desarrollo de éste estándar comenzó en Noviembre de 1996. La primera edición del estándar ECMA fue adoptado por la Asamblea General de ECMA en Junio de 1997. Dicho estándar fue enviado a ISO/IEC JTC 1 para su adopción y aprobado como estándar internacional ISO/IEC 16262, en Abril de 1998. La Asamblea General de ECMA de Junio de 1998, aprobó la segunda edición de ECMA-262 para mantenerla totalmente en consonancia con el ISO/IEC 16262. Los cambios producidos entre la primera y la segunda versión, son de carácter meramente editorial".[3]

Microsoft, que desde hace un par de años parece mostrar una voluntad decididamente más cercana a la adopción de estándares, proclama que la versión de Javascript incluida en sus navegadores (especialmente a partir de la versión 5.x), cumple rigurosamente con dicho estándar e incluso sugiere utilizar el atributo ***language = "ECMAScript"*** al construir páginas de cliente.

Javascript y el escenario de ejecución

Como cualquier otro lenguaje de *script*, **Javascript** puede utilizarse para construir páginas de servidor o de cliente. Y como es lógico, dependiendo de ese escenario de ejecución, tendrá acceso a diferentes modelos de objetos. Ya vimos el modelos de objetos de documento (DOM) que representa al navegador, en el caso del cliente.

Los *Scripts* de servidor se utilizan para realizar operaciones sobre *objetos del servidor*, tales como la ejecución de una conexión sobre una base de datos existente, una consulta SQL o un acceso a su sistema de archivos, por lo tanto el modelo de objetos variará según se trate de un servidor u otro (**Internet Information Server**, **Oracle Web Server**, **Apache Web Server**, etc.).

A diferencia de los *scripts* de cliente, los de servidor se ejecutan como si de un programa ejecutable.EXE se tratase, aunque enviando el resultado a un navegador. Pero para ello, es necesario que la página se almacene en el servidor en una carpeta ejecutable, y que este tenga instalado el "*Javascript Runtime Engine*" o motor de Javascript, que será el encargado de procesar las páginas que contengan este tipo de código. Al operar de esta forma, se devuelven al usuario ***los resultados de la ejecución del código de servidor***, y no el propio código, que no sería capaz de interpretar.

En contraste con los CGI standard (Common Gateway Interface) los programas en Javascript, están embebidos en la página, junto con el código HTML facilitando enormemente el mantenimiento de las páginas.

---

[3] "This ECMA Standard is based on several originating technologies, the most well known being JavaScript (Netscape), and JScript (Microsoft). The language was invented by Brendan Eich at Netscape and first appeared in that company's Navigator 2.0 browser. It has appeared in all subsequent browsers from Netscape and in all browsers from Microsoft starting with Internet Explorer 3.0.

The development of this Standard started in November 1996. The first edition of this ECMA Standard was adopted by the ECMA General Assembly of June 1997. That ECMA Standard was submitted to ISO/IEC JTC 1 for adoption, and approved as international standard ISO/IEC 16262, in April 1998. The ECMA General Assembly of June 1998 approved the second edition of ECMA-262 to keep it fully aligned with ISO/IEC 16262. Changes between the first and the second edition are editorial in nature."

Objetos de Java Script de servidor

Los objetos de *script* de servidor son los encargados de enlazar con fuentes de datos de cualquier tipo, almacenadas en el servidor, bien en forma de Base de datos, bien en forma de archivo de datos, que en su versión más actualizada puede adoptar el formato XML para independizarse completamente de la plataforma que lo soporta. A continuación mostramos en la Figura 28, un típico modelo de objetos de servidor.

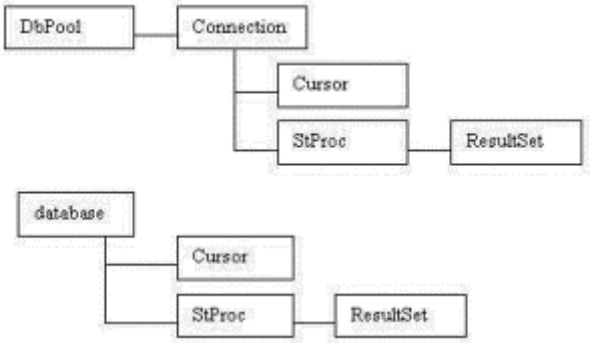

Figura 28. Modelo de objetos de Servidor de Internet

No obstante, al no ser materia de este curso vuelvo a remitir al lector a las referencias del capítulo 1º si desea obtener más información sobre este apartado. Nosotros vamos a centrarnos ahora en el script de cliente, en las características del lenguaje, y en el propio modelo de objetos de Javascript.

Características del lenguaje Javascript

Como cualquier otro lenguaje de script Javascript no genera ejecutables, sino que se basa en una librería de ***Runtime*** (tiempo de ejecución), que interpreta las sentencias según son invocadas por los mecanismos programados al efecto. No obstante, Javascript pone a disposición del programador una serie de recursos propios en forma de operadores, palabras reservadas, estructuras de control, declaración de variables, y sentencias de todo tipo, así como un conjunto de convenciones lingüísticas que deben de respetarse si se quiere codificar de acuerdo con el estándar. Vamos revisar lo más importante de sus características, y –a continuación- propondremos una serie de ejercicios típicos para que el lector pueda juzgar sus posibilidades a partir de situaciones reales, en lugar de abordar ejemplos para cada operador, estructura, palabra reservada, etc. Al lector interesado en un manual exhaustivo de Javascript podemos recomendarle "Javascript Unleashed", de R. Wagner, Ed. SAMS.

Sentencias

Un programa en Javascript es una colección de sentencias, combinadas de una forma dada para permitir la ejecución de una tarea. Una sentencia consta de una o más expresiones formadas mediante palabras reservadas, operadores, estructuras de control y otros elementos del lenguaje. En Javascript se utiliza el símbolo de punto y coma para indicar la finalización de una sentencia.

Para agrupar sentencias, utilizaremos sin embargo las llaves **{, }**, tal y como veremos en el apartado dedicado a las estructuras de control.

Operadores

Javascript define un amplio conjunto de operadores, que podemos analizar por categorías: de *comparación*, *aritméticos*, *lógicos*, de *concatenación de* cadena, de *asignación* y de *desplazamiento*.

De comparación

Son los utilizados para comparar valores, constantes o variables, siempre que estas sean del mismo tipo, esto es, semejantes. Entendiendo por tal, que -caso de no ser idénticas- puedan ser convertidas al tipo adecuado para la comparación.

Los operadores de comparación estándar de Javascript son:

- Igualdad (= =)
- Mayor que (>)
- Menor que (<)
- Mayor o igual (> =)
- Menor o igual (<=)
- Distinto (!=)

Aritméticos

Los operadores aritméticos toman una pareja de valores y devuelven uno solo. Los operadores estándar son:

- Adición (+)
- Substracción (-)
- Multiplicación (*)
- División (/)

Aparte de estos, el lenguaje aporta los siguientes operadores adicionales:

Modulo (%)

Devuelve el resto resultante de la división de los valores iniciales, lo vemos en el Código fuente 58.

```
z = x % y
20 % 6 = 2
```

Código fuente 58

Operador de Auto-incremento (++)

Este operador se utiliza para incrementar el valor de una variable en una unidad, sea el tipo que sea la base numeral utilizada.

```
x ++ //Equivale a x = x + 1
++ x //Equivale a x = x + 1, pero incrementa la variable primero,
antes de usarse
```

El operador de incremento se puede usar de dos formas diferentes cuando lo asignamos a una variable, de forma que si declaramos que Y = X++ para un valor de X = 5, primero se asignará el valor a Y, y luego se incrementará la variable X. Así, queda X = 6 e Y = 5.

Si declararemos la asignación del otro modo, Y = ++X para el mismo valor de X, primero se incrementaría la X, y después se asignaría el valor a la variable Y. Así, el resultado de la operación sería el siguiente: X = 6 e Y = 6

Operador de Decremento (--)

El decremento es semejante al operador anterior, aunque en lugar de incrementar una unidad, la resta de la variable que preceda al operador.

```
x -- //Equivale a x = x - 1
-- x //Equivale a x = x - 1, pero decrementa la variable primero,
antes de usarse
```

Al igual que el operador de incremento, en una asignación no actúa igual Y = --X que Y = X-- Ya que la primera decrementa X antes de asignar el valor a Y, mientras que la segunda asigna a Y el valor de X ya decrementado.

Negación (-)

El operador de negación precede a una variable o constante, y se utilizará para convertir el valor de una expresión en su opuesto, o para cambiar de signo un valor.

```
-x
```

Operadores lógicos

Los operadores lógicos devuelven un valor Booleano (verdadero / falso) en función de las expresiones que estos evalúen. Javascript permite los siguientes operadores lógicos:

And (&&)

El operador `&&` devuelve Verdadero cuando las dos expresiones que evalúa son verdaderas, de tal forma **`x && y`** devuelve verdadero solo cuando tanto X como Y son verdaderas. A continuación se especifican todas las situaciones.

```
V && V = V
F && V = F
V && F = F
```

```
F && F = F
```

Or (||)

El operador || es opuesto a && en tanto en cuanto a que este operador, situado entre dos expresiones, devuelve el valor booleano verdadero sólo con que una de las expresiones que evalúa sea verdadera.

A continuación, vemos las correspondencias:

```
V || V = V
V || F = V
F || V = V
F || F = F
```

Not (!)

El operador ! devuelve el valor contrario de una expresión, de forma que si la expresión es verdadera, la aplicación de este operador sobre esa variable devolverá Falso, mientras que si la expresión original es falsa, la aplicación del operador ! devolverá verdadero.

Operadores de cadenas de texto

Los operadores de cadena se ubican entre dos cadenas de texto, con la finalidad de que la operación realizada sobre ellos, devuelva una sola cadena de texto. Existen en Javascript dos operadores de cadena, cuya función es concatenar las cadenas de texto a las que se refiere. Los dos tipos de operadores de concatenación son:

- Concatenación con espacio (+)
- Concatenación sin espacio (+=)

La utilización de estos operadores es la que muestra la Tabla 6.

Instrucción	Solución
"Mi" + "Coche"	"Mi Coche"
"Mi" += "Coche"	"MiCoche"

Tabla 6

Operadores de asignación

Los operadores de asignación son los utilizados para hacer que el valor de una variable adquiera el especificado en la operación de asignación. En la Tabla 7, se especifica la lista de operadores de asignación así como sus significados.

Operador	Significado
x = y	x = y
x += y	x = x + y
x -= y	x = x - y
x *= y	x = x * y
x /= y	x = x / y
x %= y	x = x % y

Tabla 7

Operadores de desplazamiento

Estos operadores funcionan tratando sus operandos como conjuntos de bits sea cual sea la base numérica que estos utilizan, pudiendo comparar así números en base decimal con otros de otra base numeral diferente, puesto que estos operadores comparan los valores en base binaria.

Aunque la ejecución se produce a nivel binario, retornan un dato estándar Javascript. Los operadores de desplazamiento de bit, transforman sus operandos en cadenas binarias de 32 bits. Cada bit del primer operando es pareado con su correspondiente, en orden, del segundo operando. Posteriormente es aplicada la operación correspondiente a cada par de bits (uno de un operando, y otro del otro operando).

And a nivel de bit (&)

El operador &, al ser un operador de desplazamiento de bit, convierte los operandos a binario, parea los bits de un operando con los del otro, pareándolos. Entonces, aplica la operación AND estándar a esos pares de bits, tomando 1 como Verdadero, y 0 como Falso.

Como muestra la Tabla 8 se realizaría una operación AND a nivel de bit.

Operación	*Ejecución*
12 & 6	1100 & 0110 =0100

Tabla 8

Or a nivel de bit (|)

Este operador funciona como todos los operadores de desplazamiento de registro, transformando los operandos a binario, los parea por orden, y ejecuta sobre las parejas de bits la operación Or standard. Así, una operación Or a nivel de bit se ejecutaría como indica la Tabla 9.

Operación	Ejecución
12 | 6	1100 | 0110 = 1110

Tabla 9

Not a nivel de bit (~)

Al igual que los otros operadores de su clase, convierte el operando binario, y aplica sobre cada cifra binaria una operación no estándar.

Una operación no a nivel de bit resultaría como indica la Tabla 10

Operación	Ejecución
~ 12	~ 1100 = 0011

Tabla 10

Operadores especiales

En Java Script existen una serie de operadores especiales, utilizados generalmente como abreviación de código, o para trabajo directo con objetos y sus propiedades o métodos.

Operador condicional (?:)

El operador condicional es utilizado como abreviatura de código. Una condición se acompaña de dos expresiones, devolviendo la primera si la condición es verdadera, o la segunda, si la condición es falsa.

La sintaxis de este operador es la siguiente:

```
Condición ? Expresión-1 : Expresión-2
```

Que debemos entender como:

```
Si Condición entonces Expresión-1 sino Expresión-2
```

Operador Coma (,)

El operador coma, es muy simple. Separa dos o más expresiones, evaluándolas, y devolviendo el valor de la segunda expresión.

Su sintaxis es la que sigue:

```
Expresión-1, expresión-2
```

El operador Coma es utilizable para ubicar múltiples expresiones en lugares en los que solo se requiere una expresión sencilla.

delete

Este es uno de los operadores con los que trabajaremos objetos o listas. El operador Delete es utilizado para el borrado de una propiedad o un elemento de una posición específica de un array.

Existen dos sintaxis fundamentales para Delete:

- Borrado de la propiedad de un objeto:

  ```
  delete Nombre_Objeto.Propiedad
  ```

- Borrado del elemento de un Array:

  ```
  delete Nombre_Objeto [índice en el array]
  ```

new

El operador New sirve para crear una instancia de un objeto, bien sea definido por el usuario, o uno predefinido en el lenguaje. Estos objetos de lenguaje incorporan su propia función de construcción (también llamado Constructor). En definitiva, sirve para crearnos en nuestro script una réplica de un objeto ya creado, y que posteriormente personalizaremos (o no).

El modo de representación de este operador es como se muestra a continuación:

```
Nombre_Objeto = new Tipo_Objeto (Parámetros)
```

- El nombre del Objeto será el que queramos darle nosotros a la instancia que estamos creando de tal objeto.

- El tipo de Objeto se corresponderá con la función de construcción del objeto del que queremos crearnos la instancia.

- Los parámetros serán los correspondientes a la función de construcción de tal objeto.

this

El operador **this** es utilizado cuando nos queremos referir al objeto actual. Si estamos en un objeto determinado, como puede ser un control, y nos queremos referir a las propiedades de dicho control, nos referiremos al objeto actual como **this**.

De este modo, para referirnos a la propiedad o método del objeto actual, la sintaxis será:

```
this.Propiedad
```

Tipeof

El operador tipeof va seguido de una expresión, y nos devuelve una cadena de text que referencia el tipo de tal expresión. Así, si aplicamos este operador sobre una expresión nos devolverá si la expresión es un objeto, o un dato numérico, cadena de caracteres...

Su sintaxis es:

```
typeof operando
typeof (operando)
```

Los posibles valores a devolver por este operador son:

- **Object** para un objeto
- **Function** para una función
- **Number** para numerales
- **String** para cadenas de caracteres
- **Boolean** para variables booleanas

Las sentencias de Javascript son cadenas de texto pueden ocupar múltiples líneas, aunque también se da el caso de que múltiples sentencias se aúnen en una sola línea, debiendo entonces separar las sentencias con punto y coma (;).

Los Tipos de datos especiales

En Javascript, existen varios tipos de datos especiales: **Undefined**, **Null**, **Infinity** y **NaN**. No pueden utilizarse como constantes, pero hay variables con esos nombres que pueden utilizarse cuando se requieran tales valores. NaN significa (Not a Number) y se usa para aquellas operaciones matemáticas que no tienen sentido, dentro del contexto de una expresión. No hay que confundirlo con Null, que significa que tiene un significado "no hay datos" como valor en diferentes contextos, ni con undefined, utilizándose para los casos en los que el contenido de una variable no está claro en absoluto y nada –ni siquiera null- se ha llegado a almacenar. En esos casos el contenido de la variable está indefinido (undefined), aunque en muchas ocasiones se utiliza como sinónimo de Null. En el caso de que tengamos que determinar exactamente el tipo de datos almacenado, las funciones isNaN(), isFinite() y typeof(), cumplen perfectamente con esa función. No obstante, el uso de algunos de estos valores se reserva exclusivamente para las últimas versiones de los navegadores.

Un posible ejemplo de utilización sería el que muestra el Código fuente 59

```
var NaN = 0/0;
var Infinity = 1e300 * 1e300;
```

Código fuente 59

Palabras reservadas de JavaScript

Definición: Entendemos por palabras reservadas de un lenguaje aquellas que –por pertenecer a acciones definidas por el propio lenguaje- no pueden utilizarse como identificadores (para variables, constantes, etc.).

Como resumen adjuntamos dos tablas de palabras reservadas actuales y futuras, algunas de las cuales pasaremos a examinar con más detalle.

Palabras Reservadas actuales				
break	delete	function	return	Typeof
case	do	If	switch	Var
catch	else	In	this	void
continue	false	instanceof	throw	while
debugger	finally	New	true	with
default	for	Null	try	

Tabla 11

Esto en cuanto a las palabras reservadas actuales. No obstante, como las compañías productoras de software, actualizan constantemente sus productos, ya se han previsto extensiones en varias direcciones. Para darse una idea de esto, nada mejor que echar una ojeada a al cuadro de *futuras* palabras reservadas para el navegador Internet Explorer (según documentación del propio MSDN de Microsoft):

Palabras Reservadas Futuras				
Abstract	double	Goto	native	Static
Boolean	enum	implements	package	Super
Byte	export	import	private	Synchronized
Char	extends	int	protected	Throws
Class	final	interface	public	transient
Const	float	long	short	volatile

Tabla 12

Lo que se pretende con la publicación de palabras reservadas futuras no es sino prevenir a los equipos de desarrollo, para que eviten el uso de esas palabras en rutinas de programación, que pudieran ser usadas posteriormente bajo navegadores que sí las tomasen como palabras reservadas, y por tanto preservar la inversión en código fuente.

var

var es la sentencia utilizada para declarar una variable, y que, opcionalmente, permite asignarle un valor inicial. La forma de utilizar esta sentencia se especifica a continuación:

```
var nombre_variable = valor;
```

Estructuras de control

Sentencia if - else

Esta sentencia sirve para hacer que el programa decida entre una serie de operaciones u otra, en función de la evaluación de una condición. Si la condición es verdadera, se ejecuta una o varias operaciones, y si no lo es, será otra u otras las operaciones a realizar.

La sintaxis de esta sentencia se muestra a continuación:

```
If (condición) {
    acción 1;
    acción 2; }
else {
    acción 3
    acción 4; }
```

Un ejemplo práctico sería el que muestra el Código fuente 60.

```
x = 0;
if (x == 0) {
    document.write("El valor de x es 0");
}
else {
    document.write("El valor de x NO es 0");
}
```

Código fuente 60

La condición es una expresión condicional compuesta por dos operandos, y un operador condicional (Tema de operadores de Java Script), como X != 10.

switch

switch sirve para que el programa decida entre múltiples series de opciones, en función de un dato inserto por el usuario, de forma que en función del dato introducido por el usuario, el script ejecutará, bien una serie de acciones u otras, de entre un conjunto de opciones amplio.

A continuación se muestra la sintaxis de la sentencia *switch*:

```
switch(expresión)
{
        case 1 :
               operación 1;
               break;
        case 2 :
               operación 2;
               break;
        default:
```

```
                operación 3;
                break;
}
```

La última opción, no se corresponde con ninguna de las posibles introducidas por el usuario. El flujo de ejecución pasará por esa opción cuando el usuario inserte un valor no contemplada en los casos posibles. Un ejemplo de uso de switch es el que muestra el Código fuente 61

```
switch(x) {
     case 1 :
        document.write("Opción I");
        break;
     case 2 :
        document.write("Opción II");
        break;
     case 3 :
        document.write("Opción III");
        break;
      default :
        document.write("Opción por defecto");
        break;
      }
```

Código fuente 61

Si no colocásemos la sentencia Break al final de cada opción, el programa seguiría ejecutando las siguientes opciones del switch. Con Break, evitamos este problema fácilmente.

Comentarios

Los comentarios se insertan en el código del programa con la finalidad de que el programador pueda escribir cualquier cosa relevante acerca del código que considere importante. También se utiliza para marcar las diferentes zonas, con el fin de localizarlos visualmente de forma rápida.

Existen dos formas de insertar comentarios en el código, veámoslo en el Código fuente 65.

```
//Comentario de una sola línea
/*Comentario de
múltiples líneas*/
```

Código fuente 62

Bucles *while*

La sentencia **while** es utilizada con la finalidad de que el script realice una o múltiples operaciones en función de que la evaluación sea verdadera. Esto es: mientras la condición se cumpla, las operaciones que contiene se repetirán continuamente, dejando de hacerlo, en cuanto la condición se deje de cumplir.

Su sintaxis se resume en:

```
while condición
{
      acción 1
      acción 2
      acción 3
}
```

En el Código fuente 63 se muestra un ejemplo de código.

```
var x;
while (x < 10)
      {
         alert(x);
         x++;
       }
```

Código fuente 63

Bucles do - while

La sentencia **do** ... **while** se utiliza para realizar una serie de operaciones en función de una expresión, ejecutándose las operaciones de forma continua y repetida mientras la evaluación de la expresión, sea verdadero.

La sintaxis de esta sentencia es como sigue:

```
do {
      sentencia
} while (condición);
```

Vemos un ejemplo en el Código fuente 64

```
var x = 0;
do {
    x++;
    document.write(x);
while (x < 25);
```

Código fuente 64

Bucles *for*

El bucle ***for*** se utiliza con la finalidad de que el programa que realizamos realiza una o múltiples operaciones un número de veces determinado, de forma que se repita el código que contiene el bucle tantas veces como se especifique. La sintaxis de este bucle es:

```
for (expresión inicial; condición; incremento){
sentencias
}
```

En el Código fuente 65 podemos ver un ejemplo.

```
 var i;
for (i = 0;i < 10;i++) {
    document.write(i);
    if (i == 5) {
        document.write("Hola");
     }
}
```

Código fuente 65

Sentencia *break*

Esta sentencia se ubica dentro de un bucle (sentencia que permite repetir el mismo código varias veces) y se utiliza para forzar la salida de este. El bucle, se repite un número de veces determinado, o bien en función de una condición. Con la sentencia Break, podemos forzar desde dentro del propio bucle, la finalización del mismo, y el salto a la siguiente instrucción, después del bucle. Su sintaxis es la que sigue:

```
Inicio del bucle
           Instrucción 1;
           Instrucción 2;
           ....
     Break;   //Sale incondicionalemente del bucle
           Instrucción 4;
Fin del Bucle
```

Un ejemplo práctico de lo antes mencionado, es el que muestra el Código fuente 66.

```
function Prueba() {
  var i = 0;
  var x = 0;
  while (i != 6) {
         x++;
         if (x == 4) {
             break;
         }
         i++;
  }
}
```

Código fuente 66

continue

La sentencia ***continue***, al igual que la sentencia ***break***, se utiliza en un bucle (while, for), y lo que hace es terminar la ejecución del conjunto de sentencias y operaciones del bucle, pero, en contraste con la sentencia ***break***, esta no finaliza la ejecución del bucle, sino que salta la ejecución del código inserto en él, y vuelve al comienzo del bucle, para continuar su ejecución.

Un ejemplo de uso de esta sentencia es el que muestra el Código fuente 67.

```
var x = 0;
var y = 0;
while (x < 10) {
      y = y + 2;
      if (y == 8) {
            continue;
            }
      x++;
}
```

Código fuente 67

La sentencia ***continue*** puede tener un parámetro (opcional) que se corresponderá con una etiqueta. Esta etiqueta estará dentro del bucle, permitiendo que el salto se haga hasta la etiqueta, y no hasta el comienzo del bucle. En el Código fuente 68, un ejemplo de una sentencia ***continue*** con una etiqueta.

```
var x = 0;
var y = 0;
var z = 0;
while (x < 10) {
      x++;
      ETIQUETA:
      if (x 5) {
            y++;
            document.write("Mensaje " + y);
            }
      if (y 2) {
            continue ETIQUETA;
            }
      document.write(y);
}
```

Código fuente 68

delete

Se utiliza para borrar una propiedad de un objeto. Si el objeto es de tipo *array* (cadena de datos identificados mediante un índice) se utiliza para eliminar uno de los elementos del *array*. La sintaxis general de la sentencia Delete es la que se muestra a continuación:

```
delete nombre_objeto.propiedad;
delete objeto_array[índice];
```

export

Esta sentencia se utiliza para interrelacionar objetos, propiedades y funciones a otros scripts, con la finalidad de que ambos scripts compartan esos objetos, propiedades o funciones. Su sintaxis es como se muestra a continuación:

```
export nombre_objeto
export nombre_función
export nombre_objeto.propiedad
```

En el Código fuente 69 aparece un ejemplo del código.

```
export objeto1, objeto2, funcion1, funcion2;
export objeto1, objeto2.value;
```

Código fuente 69

import

La sentencia **import** tiene una finalidad semejante a la sentencia **export**. De hecho, se utilizan en conjunto, de modo que con **import**, importamos propiedades, funciones y objetos desde otro script que previamente los ha exportado con **export**.

Su sintaxis se muestra a continuación:

```
import nombre_objeto.propiedad
import nombre_función
import nombre_objeto, nombre_objeto2
```

Antes de que se pueda ordenar la importación de objetos, estos ha de haber sido exportados desde el script origen de esos objetos, cargando el script de exportación en otra ventana o marco.

function

Esta es la sentencia utilizada en Javascript para declarar una función, que ejecutará una serie de acciones cuando sea invocada en cualquier parte del script que la contiene. La función puede operar de dos maneras diferentes: Que realice una serie de operaciones o que realice una serie de operaciones, devolviendo un valor al script que invoca la función. Si la función devuelve un valor, se especifica el dato a devolver con la sentencia **return**. A continuación, se muestra la sintaxis a utilizar:

```
function nombre_función (parámetro1, parámetro 2)
{
      <Acciones>
}
```

Los parámetros son opcionales. Si no se necesitan, se pondrán igualmente los paréntesis, pero no contendrán nada. En el Código fuente 70 se muestra un ejemplo de código.

```
function F1() {
         x++;
         document.write(x);
         }
function F2() {
         x++;
         document.write(x);
         return x;
         }
```

Código fuente 70

return

Esta es la sentencia de devolución de datos de una función. Dentro de la función, que ejecuta una serie de operaciones, se puede desear que esta devuelva un valor al flujo de programa principal. Para ello se utiliza esta sentencia.

La forma de utilizarla es, dentro de una función, y de la siguiente manera:

```
function nombre_función ()
{
       Operaciones
       return valor/variable
}
```

Objetos predefinidos por Javascript

Javascript maneja los siguientes objetos predefinidos para ayudar en la construcción de rutinas complejas: ***Array, Date, String*** y ***Math***, si bien en las implementaciones actuales de navegadores, se consideran y se tratan como objetos, además, los siguientes: **Boolean**, **Function**, **Global**, **Number**, **Object**, **RegExp**, y **Error.** No obstante, los cuatro primeros, junto a **Number** se consideran objetos intrínsecos del lenguaje.

Objeto array

El objeto array se encarga de almacenar en memoria una serie de datos del mismo tipo, que más adelante podremos recuperar. Los datos que contiene el array se referencia por un número de índice, de forma que el primer elemento será **array [0]**, el siguiente será **array [1]** y así sucesivamente. El array, al ser un objeto, requiere de una construcción del mismo, antes de ser utilizado. La creación del objeto se hace utilizando su constructor, de la forma siguiente:

```
nombre_objeto = new array(10);
nombre_objeto = new array(elemento1,elemento2) ;
```

Una vez se ha creado el objeto y dado un nombre, se puede empezar a utilizar, asignando o tomando datos del mismo. Un ejemplo de utilización de array completo sería la que muestra el Código fuente 71.

```
amigos = new array(10);
amigos[0] = "David";
amigos[1] = "Roberto";
amigos[2] = "Ángel";
```

Código fuente 71

Objeto date

Es el objeto que nos permite trabajar con fechas y horas. Al ser un objeto, requiere de un constructor, semejante al utilizado por todos los objetos.(similar al objeto array, por ejemplo). Podemos crear un

objeto de tipo fecha, sin especificar parámetros, y que serviré para especificar su contenido en un punto posterior del código. También podemos crear el mismo objeto, con una fecha como parámetro. Esto es, un objeto tipo fecha inicializado con un valor de fecha. El parámetro puede recibir una fecha, o bien una fecha y una hora.

En el Código fuente 72 se muestra una construcción sin parámetros, otra con un parámetro de tipo fecha, y por último, otra más con parámetros de tipo fecha y hora. Aparecen en el ejemplo los datos que tomaría el objeto creado.

```
Dia = new Date();
Cumpleaños = new Date(74,9,25);
alert(Cumpleaños);<script>
function conFecha() { var CUMP; CUMP = new Date(74,8,25); alert(CUMP); }
function conFecha2() { var AVIS; AVIS = new Date(99,2,14,11,55,26); alert(AVIS); }
</script>
```

Código fuente 72

En el Código fuente 73, vemos cómo configurar un objeto tipo *Fecha* incluyéndole la hora, pues hemos visto en el ejemplo anterior cómo JavaScript pone por defecto la hora a 00:00:00

```
 var AVIS;
AVIS = new Date(99,2,15,11,25,0);
alert(AVIS);
```

Código fuente 73

Métodos del objeto Date

En la Tabla 13 listamos el cuadro con los métodos definidos para el objeto Date.

Métodos del objeto Date	
`getTime()`	Devuelve la hora actual
`getDate()`	Devuelve la fecha actual
`getDay()`	Devuelve el día de la semana
`getMonth()`	Devuelve el mes de la fecha especificada, entre 1 y 12
`getYear()`	Devuelve el año del objeto tipo fecha especificado

`getHours()`	Devuelve una hora entre 0 y 23
`getMinutes()`	Devuelve los minutos de la fecha especificada
`getSeconds()`	Devuelve los segundos de la fecha especificada
`setTime(hora)`	Establece la hora completa para el objeto
`setDate(fecha)`	Establece la fecha para el objeto especificado
`setDay(dia)`	Establece el día del objeto Date especificado
`setMonth(mes)`	Establece el mes del objeto tipo fecha
`setYear(año)`	Establece el año del objeto tipo fecha especificado
`setHours(hora)`	Establece la hora para le objeto
`setMinutes(minutos)`	Establece los minutos del objeto
`setSeconds(segundos)`	Establece los segundos del objeto Date

Tabla 13

En el Código fuente 74 se muestra un ejemplo práctico, en el que declaramos una fecha, y posteriormente rectificamos uno de sus parámetros (en este caso el día).

```
<script>function ponFecha()
{
 FEC=new Date(74,9,25);
            alert(FEC);
} </script>
```

Código fuente 74

Ejercicio práctico de date

En el Código fuente 75 se muestra el código necesario para ubicar la fecha en una caja de texto que exista en un formulario de la página.

```
<HTML>
<BODY BGCOLOR=DDDDDD>
<CENTER>
<FORM NAME=Formulario1>
      <INPUT TYPE=TEXT NAME=cajaTexto>
</FORM>
<SCRIPT>
      <Formulario1.CajaTexto.value=Date();
</SCRIPT>
</CENTER>
</BODY>
</HTML>
```

Código fuente 75

Obsérvese que el código del script no está dentro de una función, sino que está libre, de forma que este se ejecuta incondicionalmente. Y escribirá la fecha en la propiedad VALUE de la caja de texto que está contenida en el formulario *(Formulario1.Cajatexto1.value)*.

Objeto Math

El objeto Math, contiene propiedades y métodos para el uso de funciones y constantes matemáticas así como sus valores (en el caso de las constantes).

Propiedades de Math

Las propiedades del objeto Math contienen las constantes matemáticas de uso más frecuente:

Propiedades de Math		
E	Constante de Euler (2.718)	`y=Math.e`
LN10	Logaritmo natural de 10 (2.310)	`y=Math.LN10`
LN2	Logaritmo natural de 2 (0.693)	`y=Math.LN2`
LOG10E	Logaritmo en base 10 de E	`y=Math.LOG10E`
LOG2E	Logaritmo en base 2 de E	`y=Math.LOG2E`
PI	Relación entre la circunferencia de un circulo y su diámetro (3.14159)	`y=Math.PI`

Tabla 14

Métodos de Math

Los métodos del objeto Math se corresponden con funciones matemáticas de uso frecuente.

Métodos de Math		
max()	Recibe dos números y devuelve el mayor	y=max(x)
min()	Recibe dos número y devuelve el menor	y=min(x)
abs()	Devuelve el valor absoluto de un número	y=abs(x)
round()	Devuelve el redondeo a entero de un número	y=round(x)
pow()	Eleva x a la potencia z-ésima	y=pow(x,z)
exp()	Recibe un número y devuelve E elevado al número que recibe	y=exp(x)
sin()	Recibe un número y devuelve el seno de dicho número	y=sin(x)
cos()	Devuelve el coseno de un número	y=cos(x)
tan()	Devuelve la tangente de un número	y=tan(x)
asin()	Devuelve el arcoseno de un número	y=asin(x)
acos()	Devuelve el arcocoseno de un número	y=acos(x)
atan()	Devuelve la arcotangente de un número	y=atan(x)
log()	Devuelve el logaritmo en base E de un número que recibe	y=log(x)

Tabla 15

Ejercicio práctico de math

Con este ejercicio, se calculará el seno, coseno y tangente de 30º, y se ubicarán las respuestas en tres cajas de texto correspondientes. Veamos el Código fuente 76.

```
 <HTML>
<BODY>

<FORM NAME=Formulario2><BR>
<INPUT TYPE=TEXT NAME=CajaSeno>
<INPUT TYPE=TEXT NAME=CajaCoseno>
<INPUT TYPE=TEXT NAME=CajaTangente>
</FORM>

<SCRIPT>
    Formulario2.CajaSeno.value=Math.sin(30);
    Formulario2.CajaCoseno.value=Nath.cos(30);
    Formulario2.CajaTangente.value=Math.tan(30);
</SCRIPT>

</BODY>
</HTML>
```

Código fuente 76

Obsérvese cómo establecemos valores para las cajas de texto utilizando su propiedad *value*

Objeto string

Las cadenas de texto, en Java Script, son realmente objetos, que contienen una cadena de caracteres. Estos objetos disponen también de propiedades y métodos aplicables a la propia cadena de texto como estructura, como pueden ser su propiedad longitud (**length**), o su método convertir a cursiva (**italics()**).

El objeto String ha de ser construido previamente a su utilización. En el Código fuente 77 se muestra cómo.

```
miCadena=new String("Bienvenido a mi página web")
```

Código fuente 77

Los objetos **String** se utilizan para permitir una forma directa de manipulación dinámica de textos, sin tener que recurrir directamente a las hojas de estilo. Una vez definido el objeto String, y la cadena que contiene, el objeto es plenamente operativo.

Propiedades de String

Solo posee una propiedad: la longitud de la cadena (**length**):

```
y=cadena.length
```

Métodos de String

Métodos de String		
bold()	Fuerza que la cadena de texto aparezca en negrita	document.write(micadena.bold())
italics()	Fuerza que la cadena sea mostrada en letra cursiva	document.write(micadena.italics())
strike()	Fuerza que la cadena se muestre tachada	document.write(micadena.strike())
big()	Aumenta la fuente de la cadena como si estuviese entre etiquetas BIG de HTML	document.write(micadena.big())
small()	Hace que la cadena aparezca en fuente pequeña, como si entre etiquetas SMALL de html se encontrase	
fontcolor()	Cambia el color de la cadena de texto	document.write(micadena.fontcolor("red"))
fontsize()	Especifica el tamaño en que se mostrará la cadena (entre 1 y 7)	document.write(micadena.fontsize(3))

sub()	Hace que la cadena se muestre como un subíndice	document.write(micadena.sub())
sup()	Hace que la cadena aparezca como superíndice	document.write(micadena.sup())
fixed()	Obliga a la cadena a ser mostrada en fuente equivalente a la etiqueta TT de HTML	document.write(micadena.fixed())
blink()	Hace que la cadena de texto aparezca en la pantalla parpadeando	document.write(micadena.blink())
link()	Hace que una cadena actúe como enlace a una URL especificada	document.write(micadena.link(URL)
anchor()	Hace que el objeto string actúe como un hiperenlace a la misma página	document.write(micadena.anchor(etiqueta))
concat()	Combina el texto de dos cadenas	micadena1.concat(micadena2)
charAt()	Devuelve el carácter que está en una posición determinada de la cadena	y=micadena.charAt(8)
substr()	Devuelve una subcadena obtenida entre dos índices de la cadena principal	y=micadena.substr(3,8)
indexOf()	Devuelve la primera posición en la que aparece en la cadena una letra determinada	y=micadena.indexOf("j")
lastIndexOf()	Devuelve la última posición en la que aparece en la cadena una letra determinada	y=micadena.lastIndexOf("j")
Tolowercase()	Muestra la cadena en minúsculas	document.write(micadena.toLowerCase())
toUpperCase ()	Muestra la cadena en mayúsculas	document.write(micadena.toUpperCase())

Tabla 16

4

Prácticas con Javascript

Javascript a través de ejemplos

A continuación proponemos al lector una aproximación al lenguaje que haga acopio de todo lo estudiado hasta ahora, abordando tareas concretas dentro de la problemática actual de construcción de páginas web de cliente. A través de los ejemplos que propondremos, se trata de ver la utilidad del lenguaje, sus características de uso y sus posibilidades.

Ejercicios básicos

Construcción y llamada a una función

A modo de recordatorio, veamos la construcción básica de funciones en JavaScript, cuya misión fundamental es la de abstraer rutinas de código fuente otorgándoles un nombre, y permitiendo de esa forma su reutilización.

```
<HTML>
<HEAD>

<SCRIPT language="javascript">

function mensaje( )
{
alert( "Función de Javascript" )
```

```
}

mensaje() //Llamada a la función

</SCRIPT>
</HEAD>
<BODY>

// aquí va el código  HTML

</BODY>
</HTML>
```

Código fuente 78

La ejecución de un archivo de comandos Javascript se realiza según el denominado flujo *Top-Down*, o sea, de forma secuencial.

Utilización de cajas de diálogo

El lector recordará que hemos abordado este punto en el primer capítulo dedicado a HTML Dinámico. No obstante,

```
<HTML>
<SCRIPT LANGUAJE="JavaScript">
function RecogerDatos(BotonOpcion)
{
var coleccion, cad;
coleccion = document.all[BotonOpcion];

for (i=0;i<coleccion.length;i++) {
    if (coleccion[i].checked)
            cad = coleccion[i].value;
      }
      cad = 'Nombre: ' + txtUsuario.value + '// F.Pago: ' + cad;
      alert(cad);
}
</SCRIPT>
<BODY>
<fieldset style="width:100px">
<legend>Seleccione Forma de Pago</legend>
<INPUT type=radio name=FPago  value="Contado">Contado<br>
<INPUT type=radio name=FPago checked value="Tarjeta">Tarjeta<br>
</fieldset>
Entre nombre:
<input type="text" ID="txtUsuario" size="10">
<input type="button" name="cmdBtn" value="Ver valor"
onClick="RecogerDatos('FPago');">
</BODY>
</HTML>
```

Código fuente 79

la página muestra una selección con dos botones de radio y una caja de texto. Cuando el usuario selecciona un valor e introduce un dato en la caja de texto, al pulsar el botón cmdBtn, se visualiza el

resultado. El código fuente, realiza la acción asociando al evento ***onclick*** del botón la llamada a la función *RecogerDatos*, y pasándole como argumento el nombre del botón de radio ('Fpago').

Una vez en la función, utilizamos un bucle **for** simple para recorrer todos los botones de radio disponibles, y devolver el dato ***value*** (el equivalente a la *caption)* del control cuya propiedad ***checked*** valga verdadero. Hay que destacar la forma de acceso a la colección de botones mediante la propiedad ***all*** del objeto ***document.*** El procedimiento se puede generalizar para muchas situaciones similares.

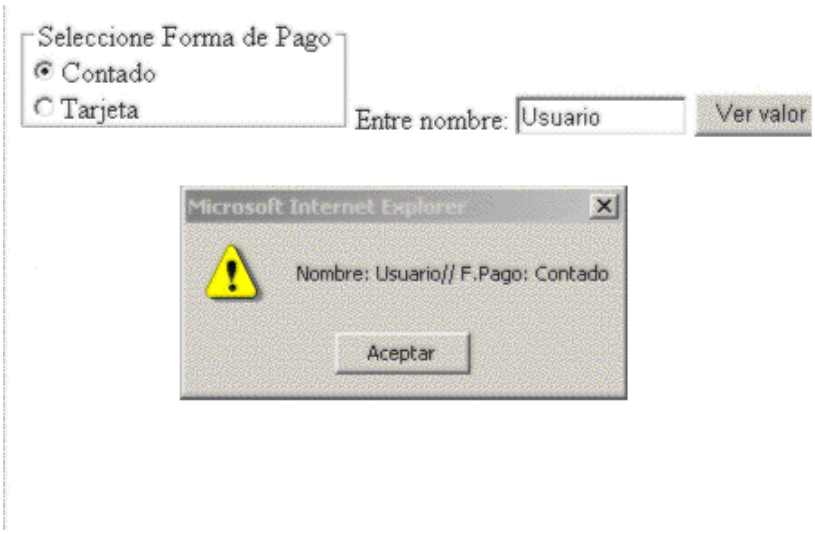

Figura 29. Salida de la página web propuesta en el ejercicio anterior

Las propiedades del objeto window

Muchas características de los sitios web actuales se gestionan a través de propiedades del objeto window, como ***location, history, URL,*** o ***appName.***. Veamos algunas de ellas relacionadas con la seguridad de acceso, la navegabilidad, la ubicación actual del documento y cómo determinar la versión y el tipo de navegador utilizado:

Control de usuarios mediante password

El Código fuente 80 efectúa una comprobación inicial al entrar en la página, preguntando al usuario su password. Dependiendo de la respuesta, lo reenvía a una página de error, o permite el acceso a una supuesta página de socios.

```
<html>
<meta http-equiv="Content-Script-Type" content="text/javascript">
<title>Control de usuarios</title>
<script language="JavaScript">
    function Password(){
      if(prompt("Clave de acceso:","") == "Leoncio"){
        window.location = "/DatosParaSocios.html";
      }else{
        window.location = "/VentanaDeError.html";
      }
    }
</script>
</head>
```

```
<body onload="Password()">
    <b>
      Debe introducir su contraseña para acceder al área de socios
    </b>
</body>
</html>
```

Código fuente 80

Hay que resaltar aquí que la propiedad que se utiliza es la propiedad ***location*** del objeto ***window***. Dependiendo el valor introducido, al asignar un valor URL válido a es propiedad, el navegador llevará a continuación a usuario a la página correspondiente.

Navegabilidad mediante la propiedad *history*

La propiedad ***history*** almacena un array con todas las direcciones a las que se ha navegado en la sesión actual. Eso, unido a las propiedades y métodos de ***history***, nos permite crear páginas en las que mediante mecanismos de redirección, podamos reenviar al usuario a la página anterior, incluso sin que nuestra página sepa a priori de cual se trata.

Mediante los métodos ***back***, ***forward*** y ***go***, podemos navegar a la página anterior, a la siguiente o ir a una página concreta del historial. También se puede conocer el número de páginas visitadas mediante la propiedad ***length***.

Ubicación actual mediante URL, y referencias agrupadas mediante *with*

La página puede conocer siempre cual es la ubicación actual mediante la propiedad URL. Además en el caso de que tengamos que hacer referencia a más de una propiedad de un objeto, la estructura de agrupación ***with***, permite hacer referencias múltiples a las propiedades de un objeto. El Código fuente 81 lo demuestra.

```
<html>
<head>
  <title>Manual EIDOS de HTML Dinámico, Modelos de objetos y Javascript</title>
</head>
<body>
  <script type="text/javascript">
    with(document){
      write("¡Hola!");
      write("<br>El título del documento es, \"" + title + "\".");
      write("<br>La URL de este documento es: " + URL);
      write("<br>¡No es necesario usar el prefijo del objeto cada vez!");
    }
  </script>
</body>
</html>
```

Código fuente 81

La estructura ***with***, permite hacer referencia a las propiedades del objeto document sin repetir el prefijo con el nombre del objeto en cada llamada. Se produce una salida donde el valor del título y la URL se toman automáticamente de las propiedades del objeto ***document***:

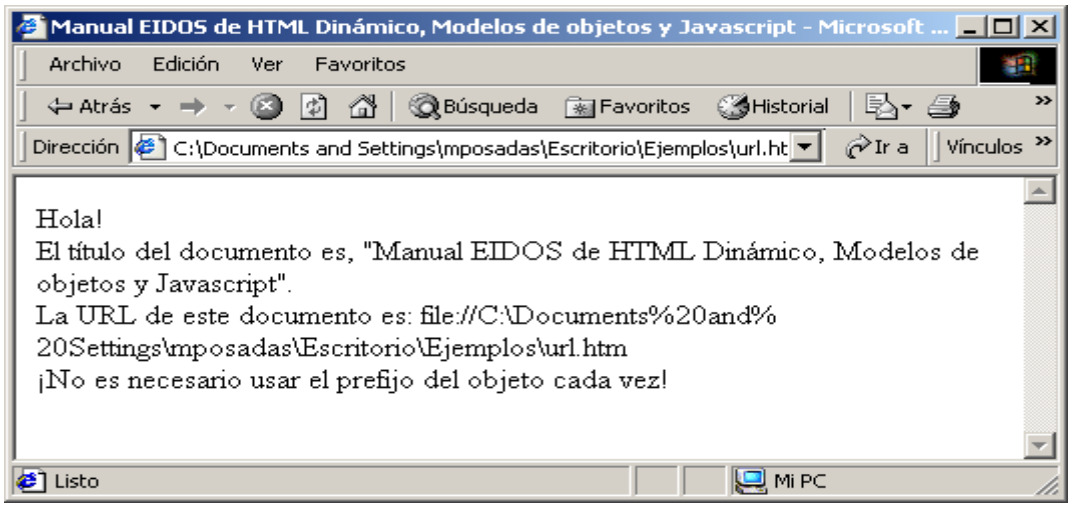

Figura 30. Salida del código anterior en Internet Explorer

Averiguar la versión y el tipo de navegador utilizado:

En numerosas ocasiones es preciso escribir código polivalente: es decir, que pueda ser usado para cualquier tipo de navegador, espcecialmente, cuando es importante que nuestras páginas sean accedidas por el mayor número de usuarios posible. Para estas situaciones, podemos utilizar las funciones provistas por el objeto ***window*** a tal efecto: ***appName, appVersion***, etc. Veamos un ejemplo de su utilización,en el Código fuente 82, mostrando la lista de propiedades disponibles.

```
<html>
<head>
<title>Manual EIDOS de HTML Dinámico, Modelos de objetos y Javascript. Marino
Posadas. (Grupo EIDOS)</title>
</head>
<body>
  <script language="JavaScript">
  <!--

    document.write("navigator.appCodeName: ".bold() + navigator.appCodeName +
"<br>");
    document.write("navigator.appName: ".bold() + navigator.appName + "<br>");
    document.write("navigator.appVersion: ".bold() + navigator.appVersion +
"<br>");
    document.write("navigator.language: ".bold() + navigator.language + "<br>");
    document.write("navigator.mimeTypes: ".bold() + navigator.mimeTypes + "<br>");
    document.write("navigator.platform: ".bold() + navigator.platform + "<br>");
    document.write("navigator.plugins: ".bold() + navigator.plugins + "<br>");
    document.write("navigator.userAgent: ".bold() + navigator.userAgent + "<br>");
    document.close();

  //--->
  </script>
</body>
</html>
```

Código fuente 82

Como vemos, el número de propiedades es suficiente para conocer en todo momento bajo qué entorno de ejecución se está viendo nuestra página. Dependiendo de esas situación podremos escribir código ejecutable para cada circunstancia. En este caso, la salida del Código fuente 82, produce una pantalla de referencias generales sobre todas las característicasque muestra la Figura 30.

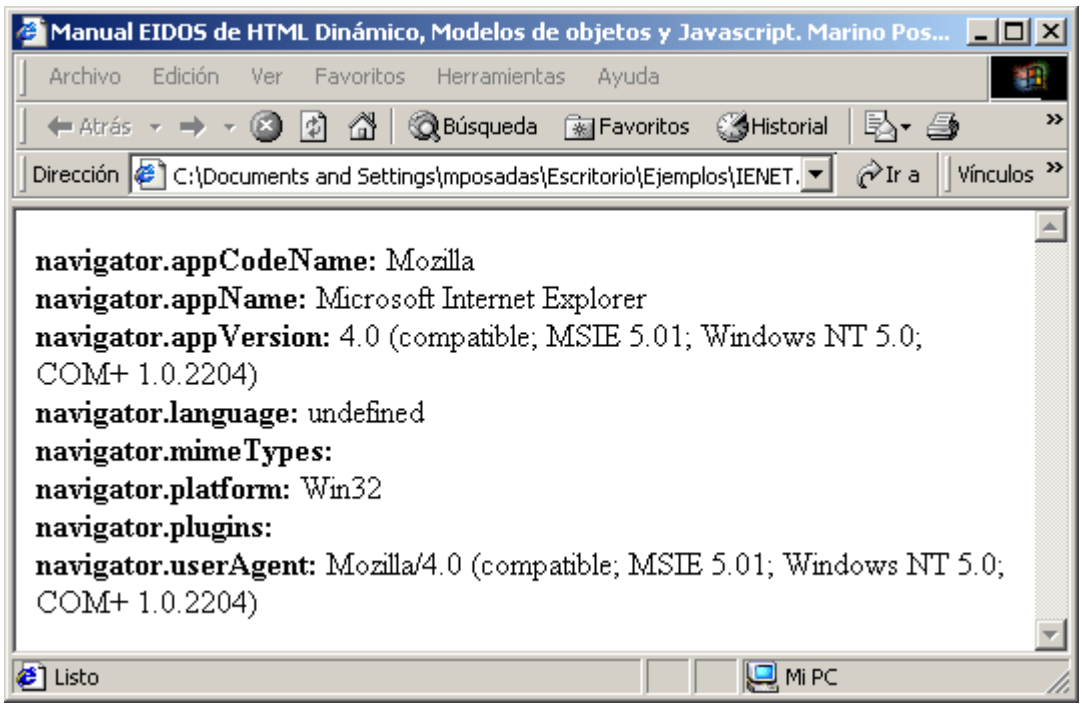

Figura 31. Página mostrando las características del navegador

Nota para programadores experimentados en orientación a objetos

A pesar de la apariencia de las relaciones clase-subclase en la jerarquía Javascript, muchos de los aspectos tradicionales de un entorno orientado a objetos no son aplicables a Javascript. En realidad la jerarquía Javascript es más una jerarquía de contenidos o de envoltorios que una jerarquía de herencia. No se heredan ni propiedades ni métodos de los objetos más altos en la cadena. No hay un mecanismo de trasvase de mensajes entre objetos en ninguna dirección. Por lo tanto, no puede invocar un método ventana mediante el envío de un mensaje a él desde un documento o plantilla. Cualquier referencia a un objeto debe ser explícita.

Los objetos Javascript predefinidos son generados sólo cuando el código HTML que contiene sus definiciones se carga en el navegador. No se puede añadir o modificar propiedades, métodos o manejadores de ningún objeto (aunque pueda añadir una propiedad a una variable que pertenece a un objeto). A lo sumo se puede modificar algunos valores de las propiedades.

Creación de un objeto

La mejor forma de verlo es con un ejemplo, lo primero que vamos hacer es crear un objeto al que le llamamos **foroEIDOS** y sus propiedades.

```
function foroEIDOS ( nNumICQ, cNombre, cApodo )
```

```
{
this.nNumICQ = nNumICQ
this.cNombre = cNombre
this.cApodo = cApodo
}
```

Código fuente 83

Con esto no creamos ninguna instancia especial. El objeto posee tres propiedades, Número de ICQ, Nombre y Apodo. Para crear una instancia necesitaremos el constructor **new.** Con ello comunicamos a Javascript que deseamos crear una nueva instancia y no ejecutar una función. Como indica el Código fuente 84.

```
userForo1 = new foroEIDOS( 4053781, "Lagarto Juancho", "Juancho" )
```

Código fuente 84

Con la instrucción que aparece en el Código fuente 85 establecemos la instancia userForo1 que tiene las propiedades.

```
userForo1.nNumICQ
userForo1.cNombre
userForo1.cApodo
```

Código fuente 85

Todas las instancias de un objeto son independientes entre sí. Por lo tanto, también podríamos asignar una instancia especial con propiedades adicionales.

```
userForo1.seccion = "Reptiles"
```

Código fuente 86

Esta propiedad especial sólo es válida para la instancia *userForo1*. Si deseamos asignar esta propiedad a todas las instancias, la definición correspondiente deberá colocarse en el constructor del objeto afectado.

```
function foroEIDOS ( nNumICQ, cNombre, cApodo, cSeccion )
{
this.nNumICQ = nNumICQ
this.cNombre = cNombre
this.cNick = cApodo
this.seccion = cSeccion
}
```

Código fuente 87

Mas aplicaciones de los objetos: Presentar Fechas y Horas

Las funciones de fecha y hora nos permiten hacer un control de los valores de reloj en Javascript, utilizando algunas de las funciones asociadas con el objeto ***Date.*** En el Código fuente 88, incluimos un ejemplo para mostrar la hora actual al cargarse una página web.

```
<html>
<head>
  <script type="text/javascript">
  <!--
    function Reloj (Horas, Minutos){
     this.Horas = Horas;
     this.Minutos = Minutos;
     this.PonerEnHora = PonerEnHora;
     this.MostrarLaHora = MostrarLaHora;
    }

    function PonerEnHora (Horas, Minutos){
     this.Horas = Horas;
     this.Minutos = Minutos;
    }

    function MostrarLaHora (){
      var line = this.Horas + ":" + this.Minutos;
      document.write ("<hr>Hora del Reloj:  " + line);
    }
  //-->
  </script>
</head>
<body>
  <script type="text/javascript">
  <!--
    var HoraActual = new Date();
    miReloj = new Reloj(HoraActual.getHours (), HoraActual.getMinutes ());
    miReloj.MostrarLaHora ();
  //-->
  </script>
</body>
</html>
```

Código fuente 88

Este documento, instancia un nuevo objeto ***Date*** llamado HoraActual y utiliza sus métodos ***getHours()*** y ***getMinutes()*** para obtener los valores que nos presenta como salida (la hora actual del sistema):

```
Hora del Reloj: 12:39
```

Otro ejemplo, más completo

En el Código fuente 89 se presenta un ejemplo más completo, accediendo a todos los recursos de fecha y hora para presentarlos como parte de la salida de una página. Se utilizan los métodos ***getMonth()***, ***getDate()*** y ***getHours()*** del objeto ***Date***.

```
<script type="text/javascript" language="JavaScript1.1">
<!--
  Date.prototype.getActualMonth = MesActual;
```

```
  Date.prototype.getActualDay = DiaActual;
  Date.prototype.getCalendarDay = LeerSemanaCalendario;
  Date.prototype.getCalendarMonth = LeerMesCalendario;

  function MesActual() {
    var n = this.getMonth();
    n += 1;
    return n;
  }

  function DiaActual() {
    var n = this.getDay();
    n += 1;
    return n;
  }

  function LeerSemanaCalendario() {
    var n = this.getDay();
    var dow = new Array(7);
    dow[0] = "Domingo";
    dow[1] = "Lunes";
    dow[2] = "Martes";
    dow[3] = "Miércoles";
    dow[4] = "Jueves";
    dow[5] = "Viernes";
    dow[6] = "Sábado";
    return dow[n];
  }

  function LeerMesCalendario() {
    var n = this.getMonth();
    var moy = new Array(12);
    moy[0] = "Enero";
    moy[1] = "Febrero";
    moy[2] = "Marzo";
    moy[3] = "Abril";
    moy[4] = "Mayo";
    moy[5] = "Junio";
    moy[6] = "Julio";
    moy[7] = "Agosto";
    moy[8] = "Septiembre";
    moy[9] = "Octubre";
    moy[10] = "Noviembre";
    moy[11] = "Diciembre";
    return moy[n];
  }

  // Comprobar los métodos creados
  var Hoy = new Date();

  document.write("<b>La presente rutina se está ejecutando en "
    + Hoy.getCalendarDay() + ", el día " + Hoy.getDate()
    + " de " + Hoy.getCalendarMonth() + " del año del Señor de "
    + Hoy.getFullYear() + " D.C. a las " + Hoy.getHours() + " horas. </b>");
//-->
</script>
```

Código fuente 89

La salida de esta página sería:

La presente rutina se está ejecutando en Miércoles, el día 31 de Enero del año del Señor de 2001 D.C. a las 13 horas.

Escritura dinámica de un documento

Una de las capacidades del objeto **document** es la de poder escribir en el búfer de la pantalla de visualización, lo que nos permite crear documentos *on-line*. En el Código fuente 90, vemos como el código que sigue puede escribir y sobrescribir en un documento, dependiendo de cómo usemos el método ***write*** del objeto **document**: Si la rutina se encuentra en el cuerpo del documento, la instrucción ***document.write*** añadirá el contenido al generado por las etiquetas, mientras que si la llamada se produce una vez cargado el documento, el método ***write*** sobrescribirá todo el búfer anulando los valores generados por cualquier etiqueta anterior.

```
<HTML>
<BODY>
Ejemplo de documento con una parte fija y otra creada dinámicamente.
<BR>
<SCRIPT LANGUAGE="JavaScript">
document.write('Parte dinámica terminada con fecha: ');
document.write(document.lastModified);
</SCRIPT>
</BODY>
</HTML>
```

Código fuente 90

En este caso la salida es:

```
Ejemplo de documento con una parte fija y otra creada dinámicamente.
Parte dinámica terminada con fecha: 01/30/2001 14:06:10
```

Esto es debido a que el script está contenido dentro del cuerpo (<BODY>) del documento, pero veamos lo que sucede si sacamos el *script* fuera de <BODY>, y añadimos un mensaje interno (Código fuente 91)

```
<HTML>
<HEAD>
  <SCRIPT LANGUAGE="JavaScript">
    function EscribirDocumento() {
document.write('Parte dinámica terminada con fecha: ');
document.write(document.lastModified);    }
  </SCRIPT>
</HEAD>
<BODY onLoad="EscribirDocumento()">
Saludos cordiales, desde el cuerpo del documento…
</BODY>
</HTML>
```

Código fuente 91

En este caso la salida anula el contenido del cuerpo de la página, y solo veremos la primera parte:

```
Parte dinámica terminada con fecha: 01/30/2001 14:12:07
```

Los saludos son sobrescritos, porque primero se ejecuta la carga del documento, y una vez terminada, se produce el evento onload, que ordena volver a escribir el búfer del documento, ignorando el mensaje de saludo.

Llamada condicional a una función

En Javascript, podemos generar estructuras complejas que se ejecuten en función de varias condiciones. Por ejemplo podemos crear funciones para el tratamiento de situaciones que dependen del navegador en el que la página se esté ejecutando. En el Código fuente 92, la función a la que se llama en el código depende del resultado combinado del operador condicional (**?:**)

```
function funcionIE() {...}

function funcionNetscape() {...}

var funcion = (IE) ? funcionIE : funcionNetscape;
  // Si IE es verdadero, funcion hace referencia a funcionIE, en caso contrario
  // funcion es una referencia a funcionNetscape
funcion();
  // La llamada que haremos realmente depende de la
  // línea anterior
```

Código fuente 92

Ejemplos de uso del modelo de objetos DOM

Uso del método *cloneNode*

Mediante el método ***cloneNode*** , podemos copiar información de otras etiquetas (elementos, desde el punto de vista de DOM), y modificar sus propiedades o añadir otras nuevas, implementando así una forma de herencia. Veamos, en el Código fuente 93, un primer ejemplo de uso de estas propiedades mediante el acceso a información de una tabla, leyendo los datos de una sola celda, y mostrándolos en una caja de diálogo.

```
<HTML>
<BODY ID="bodyNode">
     <TABLE ID="tableNode">
       <BODY>
         <TR ID="tr1Node"><TD BGCOLOR="yellow">Fila 1, Columna 1</TD>
                          <TD BGCOLOR="orange">Fila 1, Columna 2</TD>
         </TR>
         <TR ID="tr2Node"><TD BGCOLOR="yellow">Fila 2, Columna 1</TD>
                          <TD BGCOLOR="orange">Fila 2, Columna 2</TD>
         </TR>
         <TR ID="tr3Node"><TD BGCOLOR="lightgreen">Fila 3, Columna 1</TD>
                          <TD BGCOLOR="beige">Fila 3, Columna 2</TD>
         </TR>
         <TR ID="tr4Node"><TD BGCOLOR="blue">Fila 4, Columna 1</TD>
                           <TD BGCOLOR="lightblue">Fila 4, Columna 2</TD>
         </TR>
         <TR ID="tr5Node"><TD BGCOLOR="orange">Fila 5, Columna 1</TD>
                          <TD BGCOLOR="purple">Fila 5, Columna 2</TD>
         </TR>
       </TBODY>
     </TABLE>
<SCRIPT LANGUAGE="JavaScript">
tr4Obj = tr1Node.cloneNode(true);
alert( "Primer Hijo = " + tr4Obj.firstChild + "\n" +
       "Nombre del Nodo= " + tr4Obj.nodeName );
```

```
</SCRIPT>
</BODY>
</HTML>
```

Código fuente 93

Y su salida nos mostraría la tabla con diferentes colores, y la caja de mensaje a la que se llama a continuación, nada más cargarse el documento.

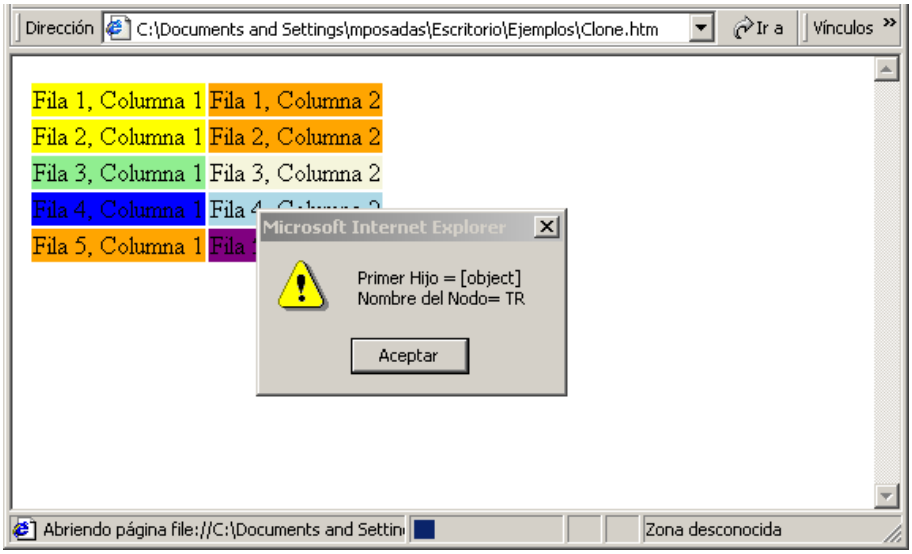

Figura 32. Acceso a las propiedades de un elemento mediante DOM

El mecanismo utilizado aquí consiste en *clonar* el contenido de una etiqueta (un objeto, en realidad, según se ha visto en el primer capítulo), para acceder después a sus propiedades y mostrarlas en una caja de mensaje.

Un poco más compleja, pero basándose en los mismos principios es la situación que sigue. Se crea una tabla totalmente a partir de código fuente de DOM, generando por código los nodos correspondientes y añadiéndolos a las colecciones del documento. Una vez terminado el proceso el mecanismo de *rendering* será el encargado de visualizar la tabla como si se hubiera construido con etiquetas HTML tipo estándar. Obtendríamos una salida en el navegador similar a la Figura 33.

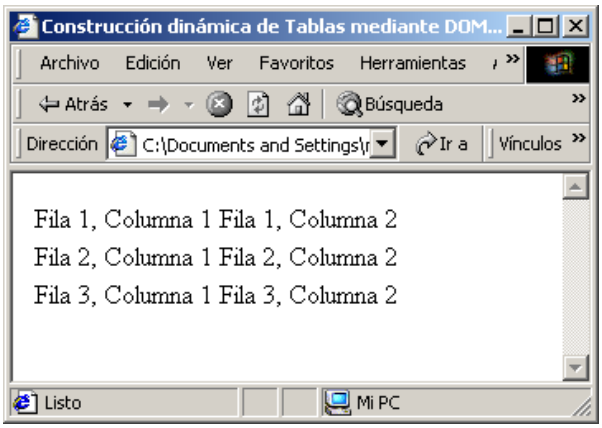

Figura 33. Construcción dinámica de tablas mediante DOM

Y el código fuente que corresponde con esta ventana es el que aparece en el Código fuente 94.

```
<HTML>
<HEAD>
<TITLE>Construcción de Tablas mediante DOM</TITLE>
</HEAD>
<BODY ID="bodyNode">
<SCRIPT LANGUAGE="JavaScript">
<!--
Fila1Columna1Obj = document.createTextNode("Fila 1, Columna 1  ");
tableObj = document.createElement("TABLE");
tbodyObj = document.createElement("TBODY");
tr1Obj = document.createElement("TR");
tr1td1Obj = document.createElement("TD");
tr1td2Obj = tr1td1Obj.cloneNode(false);
tr2td1Obj = tr1td1Obj.cloneNode(false);
tr2td2Obj = tr1td1Obj.cloneNode(false);
tr3td1Obj = tr1td1Obj.cloneNode(false);
tr3td2Obj = tr1td1Obj.cloneNode(false);
tr2Obj = tr1Obj.cloneNode(false);
tr3Obj = tr1Obj.cloneNode(false);
Fila1Columna2Obj = Fila1Columna1Obj.cloneNode(false);
Fila2Columna1Obj = Fila1Columna1Obj.cloneNode(false);
Fila2Columna2Obj = Fila1Columna1Obj.cloneNode(false);
Fila3Columna1Obj = Fila1Columna1Obj.cloneNode(false);
Fila3Columna2Obj = Fila1Columna1Obj.cloneNode(false);
Fila1Columna2Obj.nodeValue = "Fila 1, Columna 2  ";
Fila2Columna1Obj.nodeValue = "Fila 2, Columna 1  ";
Fila2Columna2Obj.nodeValue = "Fila 2, Columna 2  ";
Fila3Columna1Obj.nodeValue = "Fila 3, Columna 1  ";
Fila3Columna2Obj.nodeValue = "Fila 3, Columna 2  ";
returnValue = tableObj.insertBefore(tbodyObj);
tbodyObj.insertBefore(tr3Obj);
tbodyObj.insertBefore(tr2Obj, tr3Obj);
tbodyObj.insertBefore(tr1Obj, tr2Obj);
tr1Obj.insertBefore(tr1td2Obj);
tr1Obj.insertBefore(tr1td1Obj, tr1td2Obj);
tr2Obj.insertBefore(tr2td2Obj);
tr2Obj.insertBefore(tr2td1Obj, tr2td2Obj);
tr3Obj.insertBefore(tr3td2Obj);
tr3Obj.insertBefore(tr3td1Obj, tr3td2Obj);
tr1td2Obj.insertBefore(Fila1Columna2Obj);
tr1td1Obj.insertBefore(Fila1Columna1Obj);
tr2td2Obj.insertBefore(Fila2Columna2Obj);
tr2td1Obj.insertBefore(Fila2Columna1Obj);
tr3td2Obj.insertBefore(Fila3Columna2Obj);
tr3td1Obj.insertBefore(Fila3Columna1Obj);
bodyNode.insertBefore(tableObj);
// -->
</SCRIPT>
</BODY>
</HTML>
```

Código fuente 94

Algunos efectos visuales

Degradación de fondos

Entre los numerosos efectos visuales que se pueden conseguir mediante el nuevo modelo de objetos, uno de los más populares es la degradación del fondo de un formulario pasando de un color a otro mediante pequeños saltos de color.

El Código fuente 95 consigue este efecto mediante un bucle, combinando los colores rojo, verde y azul, aplicándolo a la propiedad ***bgColor*** del objeto ***document***:

```
<html>
<head>

  <script type="text/javascript">
  <!—
    function Degradar(InicioRojo, InicioVerde, InicioAzul, FinalRojo, FinalVerde,
FinalAzul, delay){
      for(var i = 1; i <= delay - 1; i++){
        var FinalPorcentaje = i/delay;
        var InicioPorcentaje = 1 - FinalPorcentaje;
        document.bgColor = Math.floor(InicioRojo * InicioPorcentaje + FinalRojo *
FinalPorcentaje) * 256 * 256 + Math.floor(InicioVerde * InicioPorcentaje +
FinalVerde * FinalPorcentaje) * 256 + Math.floor(InicioAzul * InicioPorcentaje +
FinalAzul * FinalPorcentaje );       }
    }
  //-->
  </script>

</head>
<body>
  <script type="text/javascript">
  <!—
   Degradar( 255,255,240,0,128,128,170 );
  //-->
  </script>

</body>
</html>
```

Código fuente 95

Al mostrarse la página comienza con color clarito en verde que va volviéndose verde oscuro según se van aplicando los valores generados por el bucle a la propiedad ***bgcolor***.

Posicionamiento por capas mediante estilos

Aunque ya se ha tratado el tema en el capítulo 2º, el código siguiente muestra un ejemplo sencillo de posicionamiento por capas utilizando estilos definidos en la misma página.

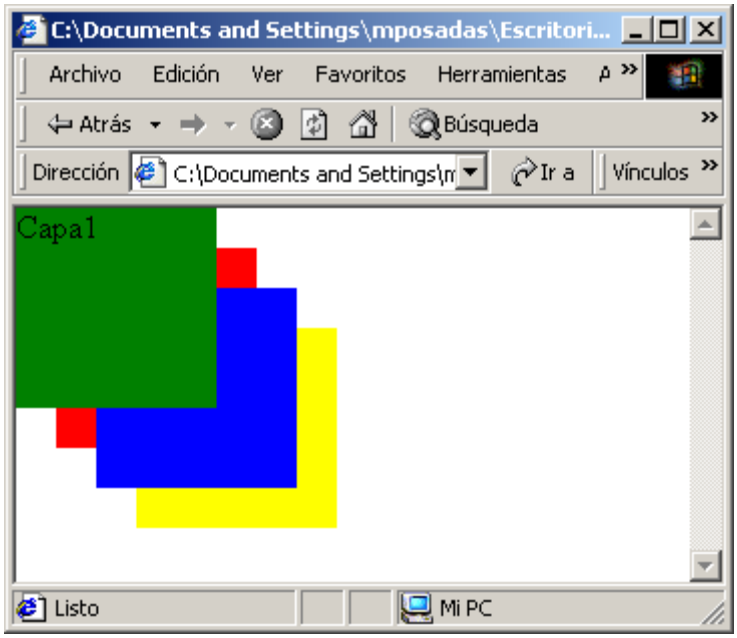

Figura 34. Posicionamiento por capas mediante estilos

Efecto que se consigue mediante el Código fuente 96

```
<html>
<head>
  <style type="text/css">
  <!--
  #Primero{
  background-color: green;
  height: 100;
  left: 0;
  position: absolute;
  top: 0;
  width: 100;
  z-index: 2;
  }

  #Segundo{
  background-color: red;
  height: 100;
  left: 20;
  position: absolute;
  top: 20;
  width: 100;
  }

  #Tercero{
  background-color: blue;
  height: 100;
  left: 40;
  position: absolute;
  top: 40;
  width: 100;
  z-index: 1;
  }

  #Cuarto{
  background-color: yellow;
  height: 100;
```

```
  left: 60;
  position: absolute;
  top: 60;
  width: 100;
  }

  -->
  </style>
</head>
<body>
  <p name="Capa1" id="Primero">
    Capa1
  </p>
  <p name="Capa2" id="Segundo">
    Capa2
  </p>
  <p name="Capa3" id="Tercero">
    Capa3
  </p>
  <p name="Capa4" id="Cuarto">
    Capa4
  </p>
</body>
</html>
```

Código fuente 96

Para más información recomendamos al lector algunos sitios web, donde podrá encontrar interesante información relacionada con funciones avanzadas del HTML Dinámico.

Algunos sitios web de utilidad para los lectores

http://msdn.microsoft.com/msdnmag

http://www.microsoft.com/mind/default.asp

http://www.microsoft.com/com/

http://www.microsoft.com/com/tech/dcom.asp

http://www.microsoft.com/com/tech/activex.asp

http://www.vbthunder.com/

http://www.insteptech.com/VBTipsFr.htm

http://www.sourcesite.simplenet.com/visualbasic/

http://www.vbinformation.com/

http://www.programando.com/visualbasic/mundovisual/

http://vbprogzone.cjb.net/

http://www3.btwebworld.com/choruscomputing/vb_expert/home.htm

LaLibreríaDigital.com Libros en formato digital para profesionales de la informática

Si quiere ver más textos en este formato, visítenos en: http://www.lalibreriadigital.com.
Este libro tiene soporte de formación virtual a través de Internet, con un profesor a su disposición, tutorías, exámenes y un completo plan formativo con otros textos. Si desea inscribirse en alguno de nuestros cursos o más información visite nuestro campus virtual en: http://www.almagesto.com.

Si quiere información más precisa de las nuevas técnicas de programación puede suscribirse gratuitamente a nuestra revista ***Algoritmo*** en: http://www.algoritmodigital.com. No deje de visitar nuestra reviata ***Alquimia*** en http://www.eidos.es/alquimia donde podrá encontrar artículos sobre tecnologías de la sociedad del conocimiento.

Si quiere hacer algún comentario, sugerencia, o tiene cualquier tipo de problema, envíelo a la dirección de correo electrónico lalibreriadigital@eidos.es.

© Grupo EIDOS
http://www.eidos.es