

Diseño de Páginas Web

Más allá de los tags...

Versión 1.1

Por Fco. José Hurtado Mayén
www.francisco.hurtado.com

Carta al lector

Elaboré este manual para la realización del curso de HTML que ya he impartido dos veces con la Junior Empresa de la Facultad de Informática de la Universidad de Murcia, INFOMUN (<http://www.um.es/infomun>).

Como podréis comprobar, el manual ha sido realizado incluyendo muchas fuentes diferentes y ha sufrido muchas modificaciones, algunas de las cuales han podido no realizarse correctamente, por lo que agradeceré que me hagáis llegar cualquier comentario o sugerencia desde mi página web personal www.francisco.hurtado.com

Bueno, espero que disfrutéis y encontréis todo lo que buskais en este manual, y que en caso de dudas, no tengáis problema en enviarme un mensaje para que pueda solventar vuestras dudas y así mejorar las próximas ediciones del manual.



¿Porqué este manual es gratis?

Este manual ha requerido muchas horas de trabajo por mi parte, y es además, parte de un material educativo que empleo para el curso de Diseño de páginas web.

Por eso este manual no debería ser gratis, porque ha requerido muchas horas de trabajo por mi parte... Por otra parte, he pensado que todos tenemos derecho a tener un manual medio decente de páginas web y tampoco está tan bien el mío... ¿no crees?

Bueno, si crees que mi trabajo se merece algún incentivo, ya sea en metálico (lo que sea) o en especie (bocadillos, tortillas de patatas, discos compactos, libros, etc), puedes enviarmelo a mi Facultad a la dirección:

Fco. José Hurtado Mayén
Delegación de Alumnos
Facultad de Informática
Universidad de Murcia
30100 – Espinardo
Murcia (España)

En fin... espero contar con tu colaboración y ayuda.

Disfruta este manual,

Fco. José Hurtado Mayén
www.francisco.hurtado.com

Diseño de páginas web

- A. Direccionamiento en Internet.
- B. Documentos HTML.
- C. Gráficos.
- D. Javascript.
- E. Java.
- F. ActiveX.
- G. Streaming.
- H. Tecnología Push.
- I. HTML Dinámico.

A. Direccionamiento en Internet

1. Números IP y DNS.
2. Programas de aplicación y sus protocolos.
3. La Multipurpose Internet Mail Extension (MIME).
4. Los Uniform Resource Location (URL).

A. Direccionamiento en Internet

La problemática del direccionamiento fue el aspecto principal que se tuvo en cuenta al crear los protocolos fundamentales TCP/IP. Efectivamente, Internet se caracteriza más por la utilización de estos protocolos de transmisión que por toda la cantidad de hardware (cables, enrutadores, etc) utilizado para la conexión de ordenadores. En la mayoría de las redes locales, junto al protocolo TCP/IP se utilizan también otros protocolos de red, por ejemplo, el IPX de Novell Netware. Por lo tanto no es acertado considerar el hardware de red como componente directa de Internet. Por eso es imposible desarrollar en Internet un concepto de direccionamiento basado directamente en el hardware, tal como sucede, por ejemplo, con el teléfono normal donde, por regla general, cada número de teléfono tiene asignado un solo extremo de línea.

1. Números IP y DNS.

El Internet Protocol (IP) define la forma de los paquetes de datos que se transmiten por la red. Cada uno de estos paquetes debe llevar la dirección del remitente y la del destinatario en forma de número IP. Cada número IP corresponde unívocamente a una de las unidades activas conectadas a Internet. Estas unidades pueden ser tanto ordenadores con aplicaciones de cliente o de servidor, como enrutadores utilizados para la conexión de los diferentes segmentos de la red.

Un número IP se compone de una serie de cuatro números de un byte de tamaño cada uno y que pueden tomar cualquier valor entre 0 y 255. Por lo general los cuatro números que forman un número IP van separados entre sí por un punto. Los números IP son un reflejo directo de la división de la Internet en diferentes redes locales (LAN). Estas redes locales se conocen en Internet como "subredes" y pueden identificarse en cada caso por los dos o tres primeros números de la dirección IP. Esta división de la dirección IP en dirección del ordenador y dirección de la subred es una condición importante para que el enrutador (el ordenador responsable de enviar los paquetes de datos por el buen camino desde cada nodo de la red) pueda desarrollar su trabajo de manera efectiva.

Desde el punto de vista técnico, estas series de números que forman el IP resultan muy apropiadas para el tráfico de datos. Pero para la comodidad del usuario representan más bien un impedimento. Por eso, paralelamente al sistema de números IP, se ha creado un sistema de nombres que permite identificar con palabras a cada ordenador. Este sistema es el Domain Name System, abreviado DNS. Un nombre de ordenador dentro del DNS consta de varias partes separadas entre sí

por puntos. Estas partes del nombre de dominio están organizadas jerárquicamente de modo que las partes del nombre que se encuentran más a la derecha responden a indicaciones cada vez más generales. La última parte del nombre hace referencia a la nación o, en el caso de los ordenadores de EE.UU., a una de las seis áreas que se especifican en la siguiente tabla:

<i>Nombre del área</i>	<i>Significado</i>
Com	“Commercial” Ordenadores administrados por empresas comerciales.
Edu	“Educational” Ordenadores administrados por escuelas y universidades.
Gov	“Government” Ordenadores administrados por organismos gubernamentales.
Mil	“Military” Ordenadores administrados por organismos militares.
Org	“Organization” Ordenadores administrados por organizaciones privadas sin fines lucrativos.
Net	“Network” Ordenadores administrados por organizaciones que se encargan de la administración o la organización de redes de ordenadores.

Un nombre típico de dominio podría ser el siguiente:

cyber.mecheng.mich.edu

El ordenador con este nombre de dominio sería un ordenador bautizado por su usuario con el nombre de “Cyber”, que se encuentra dentro de la red local del área especializada “ingeniería mecánica” y que, a su vez, forma parte de la red de la Universidad de Michigan, incluida finalmente en el área “Educación”. Un ejemplo de nombre de dominio de un ordenador español sería:

home.servicom.es

En este caso, se trata del ordenador en que se encuentran las “homepages” principales de Servicom.

Lo importante es que los nombres de ordenador del DNS puedan ser traducidos a los correspondientes números IP. Para esta función existen en Internet ordenadores especiales: los servidores. Así, cuando un usuario escribe un nombre DNS en el correspondiente programa cliente de Internet, el ordenador debe establecer en primer lugar una

conexión con uno de los servidores de nombres y consultar la dirección IP correspondiente. El usuario no puede establecer la conexión deseada hasta que no se conoce esta dirección.

2. Programas de aplicación y sus protocolos

Los protocolos TCP/IP son los que hacen posible la transmisión de datos en forma de paquetes de datos. Para el usuario esta información resulta de escaso interés, pues está interesado principalmente en cargar y enviar unidades mayores de información, como archivos, correo electrónico o páginas completas de la web. La tarea de dividir estos bloques de información relevantes para el usuario en bloques más pequeños, compatibles con los paquetes IP, y reagruparlos nuevamente una vez realizada la transmisión, corre a cargo de programas especiales. Estos programas especiales se dividen básicamente en dos grandes grupos: clientes y servidores. Los programas cliente son los que utiliza el usuario para su comunicación a través de Internet. Puede diferenciarse dentro de este grupo entre clientes de correo electrónico, clientes de news (conocidos como "readers"), clientes Gopher y programas FTP. En el contexto que nos ocupan desempeñan un papel esencial los browsers WWW, incluidos también en el grupo de programas clientes. Por otra parte, los programas servidores son los que mantienen a punto la información y la guardan temporalmente.

En Internet se desarrollan paralelamente una gran cantidad de servicios. Actualmente es necesario instalar en un sistema informático varios programas, con sus respectivos protocolos, para disfrutar de la diversidad de servicios en Internet. Los diferentes protocolos necesarios para los programas se diferencian en el número de puerto, el cual es a su vez una parte integrante del URL correspondiente. La siguiente tabla recoge los protocolos utilizados con mayor frecuencia en Internet:

Protocolo	Puerto estándar	Explicación
HTTP	80	El "Hypertext Transfer Protocol" es el protocolo creado para la transmisión de archivos HTML.
FTP	21	El "File Transfer Protocol" sirve para copiar archivos entre ordenadores de la red.
Telnet	23	El protocolo Telnet permite el pleno acceso (un log-in) a un ordenador remoto. A diferencia de FTP, con este protocolo se

Protocolo	Puerto estándar	Explicación
		pueden iniciar programas en el ordenador remoto.
SMTP	25	El "Simple Mail Protocol" se utiliza para comunicar entre sí servidores de correo electrónico.
POP2	109	El "Post Office Protocol" permite el acceso a los buzones electrónicos.
POP3	110	POP3 es una nueva variante de POP2.
NNTP	119	El "Network News Protocol" se utiliza para la transmisión de noticias de USENET-News.
Gopher	70	Gopher es el predecesor, orientado a menús, de HTTP y de la WWW en general.

Hay que añadir que los browsers de la WWW no sólo comprenden el protocolo HTTP, fundamento de la WWW, sino que pueden comunicarse con servidores de los demás servicios de Internet a través de otros protocolos. Esta "universalidad" es uno de los factores básicos para el rendimiento en general de la WWW.

3. La Multipurpose Internet Mail Extension (MIME)

Si se utilizan correctamente los métodos explicados en los apartados anteriores para el direccionamiento de los ordenadores de Internet y la especificación en cada caso del protocolo de transmisión adecuado, no hay ningún obstáculo para la transmisión de datos dentro de Internet. Es decir, es posible solicitar archivos de cualquiera de los ordenadores de la red repartidos por todo el planeta.

Pero lo realmente interesante es, claro está, que el programa que proporciona los archivos pueda ofrecerle al usuario inmediatamente la información contenida en ellos. Ese es precisamente el comportamiento que se espera de cualquier browser de la web. Los textos se presentan en el formato adecuado, las imágenes se muestran a color y los documentos de sonido se reproducen a través de una tarjeta de sonido (siempre que el usuario tenga una instalada). Para que todo eso pueda

ser así, el browser debe disponer en cada caso de la información necesaria sobre el tipo de archivo. Esta información se oculta en el nombre del archivo, exactamente en la extensión del mismo. El sistema utilizado para elaborar un estándar con el significado de las extensiones de archivo se denomina “Multipurpose Internet Mail Extensions”, abreviado MIME.

Las especificaciones MIME (por ejemplo, RFC1521 y RFC1522) dictaminan que cada tipo de archivo se especifique mediante dos palabras clave separadas entre sí por una barra transversal “/”. La primera de estas palabras es el “Content Type” y asigna cada archivo a uno de los siete grupos predeterminados:

Content Type	Significado
Text	Documento de texto
Multipart	Documento que consta de varias partes diferentes.
Message	Mensaje
Image	Documento de imágenes.
Audio	Documento de sonido.
Video	Vídeo.
Application	Aplicación (se utiliza también para documentos que sólo pueden ser utilizados mediante aplicaciones especiales).

La segunda palabra clave especifica el tipo exacto de archivo. Así, los documentos HTML, por ejemplo, pertenecen al tipo MIME “text/html”, mientras que los textos en puro ASCII pertenecen al tipo “text/plain”.

Lo decisivo ahora es que el programa cliente y servidor sepa reconocer por la extensión de archivo a qué tipo pertenece un archivo. Para ello existen archivos especiales de traducción que suelen llevar un nombre del tipo “mime.typ”. En la tabla siguiente se indican algunas de estas traducciones.

Tipo MIME	Extensiones correspondientes
Application/msword	Doc
Application/octet-stream	Bin
Application/postscript	Ai eps ps
Application/zip	Zip
Audio/basic	Au snd
Audio/wav	Wav
Image/gif	Gif
Image/jpg	Jpeg jpg jpe
Image/tiff	Tiff tif
Text/html	Html htm
Text/plain	Txt
Text/richtext	Rtx
Video/mpeg	Mpeg mpg mpe

Tipo MIME	Extensiones correspondientes
Video/quicktime	Qt mov
Video/msvideo	Avi

4. Los Uniform Resource Locators (URL)

En los documentos HTML las direcciones se indican como valores de atributos de los diferentes tags HTML. El nombre de estos atributos puede ser, por ejemplo, SRC o HREF. Los valores de los atributos son variables de carácter que representan lo que se denomina URL.

Un URL ("Uniform Resource Locator") se compone de varias partes que guardan una relación entre sí variable según correspondan a un tipo de protocolo o a otro. Sin embargo, de manera general puede decirse que un URL consta de una serie de cinco componentes como máximo:

- Indicación del protocolo de la aplicación (simultáneamente palabra clave respecto a la sintaxis).
- Indicación de una dirección del ordenador remoto (host).
- Puerto de comunicación a emplear.
- Ruta de acceso y nombre de archivo.
- Otros datos.

Todos estos componentes deben estar separados entre sí por algún carácter o conjunto de caracteres. El segundo componente del URL (dirección del host) puede indicarse tanto en forma de número IP como de nombre dominio. A veces puede ser obligatorio completar la dirección del host indicando también el número de puerto y el nombre de usuario con la correspondiente contraseña.

En los siguientes componentes se especifican las particularidades sintácticas de los URL relacionados con los protocolos más importantes.

La URL de un documento ofrecido por un servidor HTTP empieza siempre indicando el protocolo que se ha de utilizar, esto es, "HTTP". A continuación de este dato y separada por una doble barra transversal "/" se escribe la dirección del ordenador. Con el protocolo HTTP en principio consta simplemente del número IP o el nombre de dominio. Si se desea se puede indicar también un número de puerto, separado de la dirección en sí del ordenador por dos puntos ":". Si se omite este último dato, se utiliza el puerto estándar del protocolo HTTP, es decir, el puerto 80. El siguiente componente de la dirección está formado por la ruta de acceso y el nombre de archivo que se solicita por HTTP.

La ruta se escribirá separada de la dirección del ordenador por una barra transversal "/". Los nombres de los directorios de la ruta también se deben escribir separados por una barra transversal "/" y no por el conocido backlash o barra invertida de MS-DOS: "\". Si se omite la ruta

de acceso puede suceder que como respuesta a la consulta se obtenga un documento estándar. Por último, el URL de HTTP puede contener otros datos que o bien se han de transmitir a la dirección de destino o bien son indicaciones de posición para el browser. En este último caso se trata del nombre de un ancla en el documento solicitado. Este nombre debe ir separado del resto del URL por el carácter especial "#".

No se transmite al servidor, ya que sólo es importante para decidir qué parte del documento debe ser presentada por el browser. También es posible indicar datos destinados a programas CGI que se desarrollen en el servidor. Este tipo de informaciones deben ir separadas del resto del URL con un signo de interrogación.

Con browsers WWW se puede acceder a informaciones transmitidas con otros protocolos diferentes de HTTP. La tabla siguiente contiene la descripción de algunos URL permitidos en la WWW.

Nombre	Ejemplo-URL
FTP	ftp://user:password@ftp.serc.es/dir/prg.exe
FILE	File:///unid:\directorios\fichero.ext
Gopher	Gopher://goph.edu
mailto	mailto:user@host.es
NNTP	nntp://host.es/group
telnet	telnet://user:password@host.es

B. Diseño de documentos con HTML

1. Nociones fundamentales sobre la sintaxis HTML
2. Documentos sencillos.
3. Listas.
4. Imágenes y enlaces.
5. Formularios.
6. Tablas.
7. Mapas de imágenes.
8. Frames (marcos).
9. Meta elementos.
10. El futuro de HTML.

B. Diseño de documentos con HTML

Los documentos fundamentales de la WWW son los hipertextos. Los archivos correspondientes tienen la extensión HTML o HTM. La especificación HTML determina que estos archivos deben ser textos ASCII de 7 bits. Por eso no es posible "esconder" las informaciones que sirven para controlar el diseño del documento en caracteres no representables. Observando un documento HTML con un editor normal se puede ver tanto el contenido del documento como los comandos de formato. La sintaxis exacta de estos comandos de formato se define en las "Document Type Definitions" (DTD), basadas en el lenguaje de declaración SGML. No obstante, hay que hacer hincapié en que todos los trabajos de la definición de este tipo de normas siguen, en general, el desarrollo de los programas de casas como Netscape y Microsoft, de modo que la auténtica realidad de HTML no puede estar recogida totalmente en los documentos DTD existentes. Puesto que una exposición detallada de esta problemática no es posible dentro del marco de este capítulo de introducción, si te interesa más a fondo el tema, puedes consultar la bibliografía especializada sobre el tema HTML, así como las páginas web del Consorcio de las tres W¹.

1. Nociones fundamentales sobre la sintaxis HTML

Los comandos HTML se encuentran en los documentos (que, como ya se ha dicho, constan de puro texto ASCII de 7 bits) separados del texto por paréntesis angulares. La expresión que se encuentra entre estos paréntesis, es decir, entre el signo de "menor que" (<) y el signo de "mayor que" (>) se denomina tag HTML. El tag en sí consta de una palabra clave y eventualmente, otras informaciones que forman los atributos del tag.

Los tags, que sirven para el diseño de las diferentes partes del texto, suelen aparecer por parejas. En ese caso, se diferencia entre el tag inicial y el tag final correspondiente que contiene la misma palabra clave que el tag inicial pero precedida por el carácter "/". En los tags finales no tiene sentido introducir atributos y por lo mismo, no están previstos en ellos. Cada elemento HTML formado por un tag inicial y un tag final del modo descrito se denomina contáiner, ya que puede contener texto y otros elementos HTML. Las definiciones sintácticas oficiales de HTML prevén que en algunos containers pueda suprimirse el tag final. Esta medida resulta especialmente adecuada para aquellos elementos que finalizan implícitamente con el inicio de un elemento similar, por ejemplo en los párrafos <P> o en las entradas de lista .

¹ <http://www.w3c.org>

**Además del contáiner, HTML cuenta con otros elementos que aparecen siempre en solitario, es decir, sin tag final y que, por lo mismo, no pueden contener texto ni otros elementos HTML. Este es el caso de los saltos de línea
 y de las imágenes vinculadas .**

Los contáiners HTML deben formar un mosaico coherente. Así no está permitido seleccionar una entrada de lista con si el texto correspondiente no forma parte de una lista que no pueda ser enmarcada, por ejemplo, con ... o <LO>...</LO>. Las normas para el correcto anidamiento se encuentra asimismo en los documentos DTD. En los apartados siguientes se verán los aspectos fundamentales a través de ejemplos típicos.

2. Documentos sencillos

El marco externo de un documento HTML es siempre un elemento HTML que contiene un HEAD y un BODY. De ese modo la cabecera del documento, que al igual que el elemento TITLE puede contener otras informaciones no representables, queda separada del cuerpo del mismo. El contenido del elemento TITLE no se muestra como parte del documento sino que aparece en la barra de títulos de la ventana principal de Internet Explorer o Netscape Navigator.

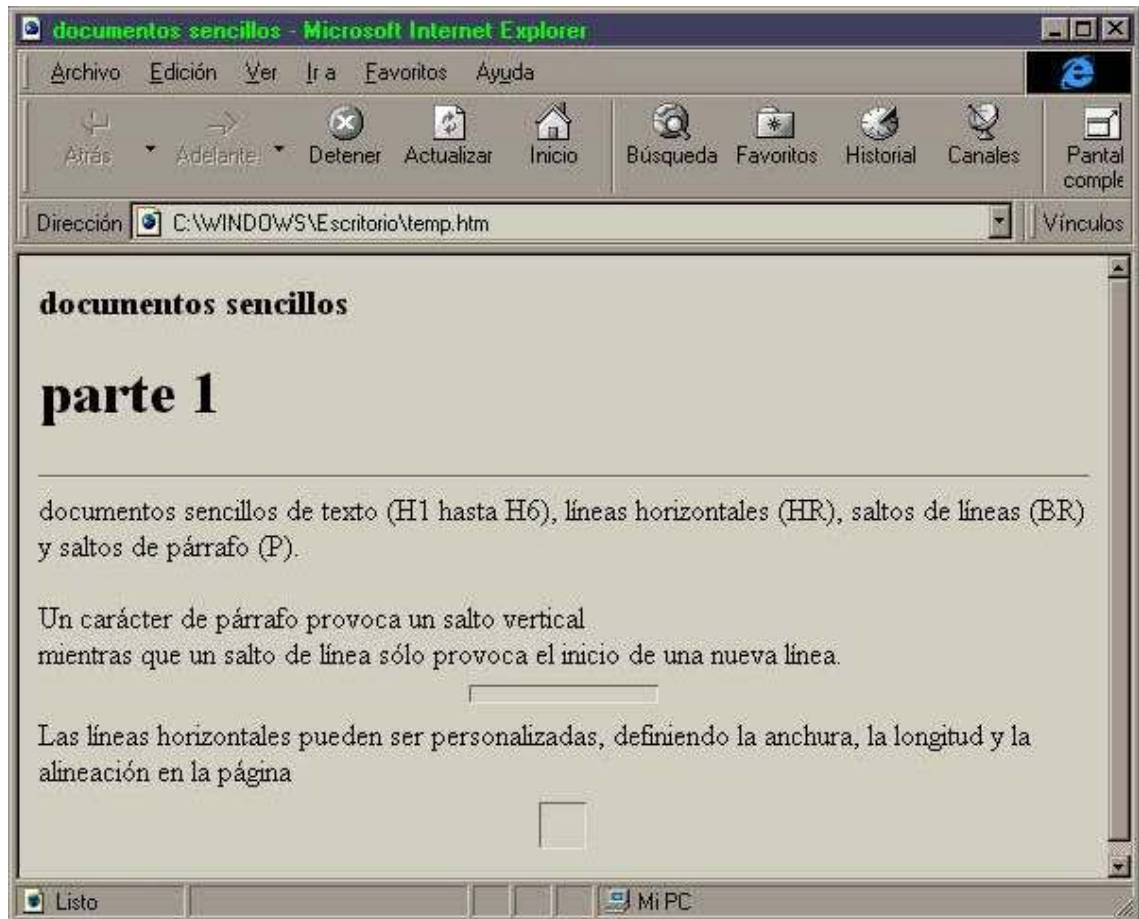
**Se pueden seleccionar seis tamaños diferentes de títulos. Estos tamaños corresponden a los elementos H1 a H6. Con el elemento HR se crea una línea horizontal. Los diferentes párrafos van señalados con el elemento P. Finalmente, con el elemento
 se pueden forzar saltos de línea.**

El uso de los comandos básicos HTML, que se acaba de mencionar queda ilustrado en el siguiente documento de ejemplo:

```
<HTML>
<HEAD>
<TITLE>documentos sencillos</TITLE>
</HEAD>
<BODY>
<H3>documentos sencillos</H3>
<H1>parte 1</H1>
<HR>
documentos sencillos de texto
(H1 hasta H6),
líneas horizontales (HR),
saltos de líneas (BR)
y saltos de párrafo (P).
<P>
Un carácter de párrafo provoca un salto vertical
<BR>
mientras que un salto de línea sólo provoca el inicio de una nueva
línea.
<HR WIDTH="100" SIZE="10" ALIGN="CENTER">
Las líneas horizontales pueden ser personalizadas, definiendo la
anchura, la longitud y la alineación en la página
```

```
<HR WIDTH="25" SIZE="25" ALIGN="CENTER">
</BODY>
</HTML>
```

Internet Explorer mostrará este documento tal como sigue:



Hay un a gran cantidad de tags para el diseño de la fuente. El siguiente ejemplo contiene los comandos más importantes de este grupo.

```
<HTML>
<HEAD>
<TITLE>Sin título</TITLE>
</HEAD>
<BODY>
<H3>Documentos sencillos</H3>
<H1>Parte 2</H1>
<HR>
Segmentos de textos individuales pueden resaltarse con escritura en
<B>negrita</B> o <I>cursiva</I>.
<P>
También es posible <SMALL>reducir</SMALL> o <BIG>aumentar</BIG> el
tipo de letra.
<P>
También pueden escribirse <SUB>subíndices</SUB> y
<SUP>superíndices</SUP>
<P>
<CENTER>
```

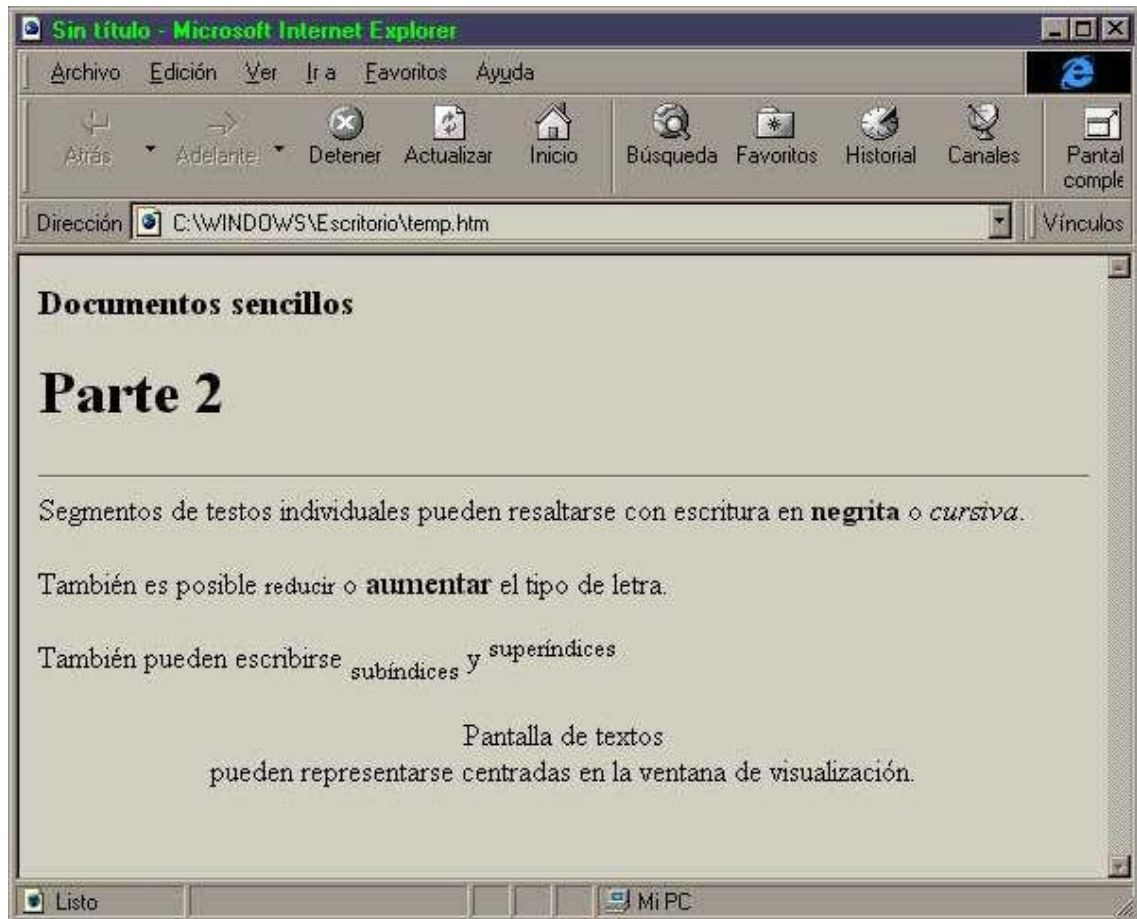
Pantalla de textos

pueden representarse centradas en la ventana de visualización.

</CENTER>

</BODY>

</HTML>



En la tabla siguiente se muestran todos los tags HTML que sirven para diseñar la presentación de la fuente. Algunos de los tags expuestos designan principalmente el contenido del textos seleccionado. La manera en que se representa el texto seleccionado puede variar ligeramente de un browser a otro.

Tag	Significado
U	Texto subrayado
S	Texto tachado
TT	Fuente no proporcional.
I	Cursiva
BIG	Texto en letra grande
SMALL	Texto en letra pequeña
B	Negrita
EM	Realzado
STRONG	Muy realzado
CODE	Código de programa

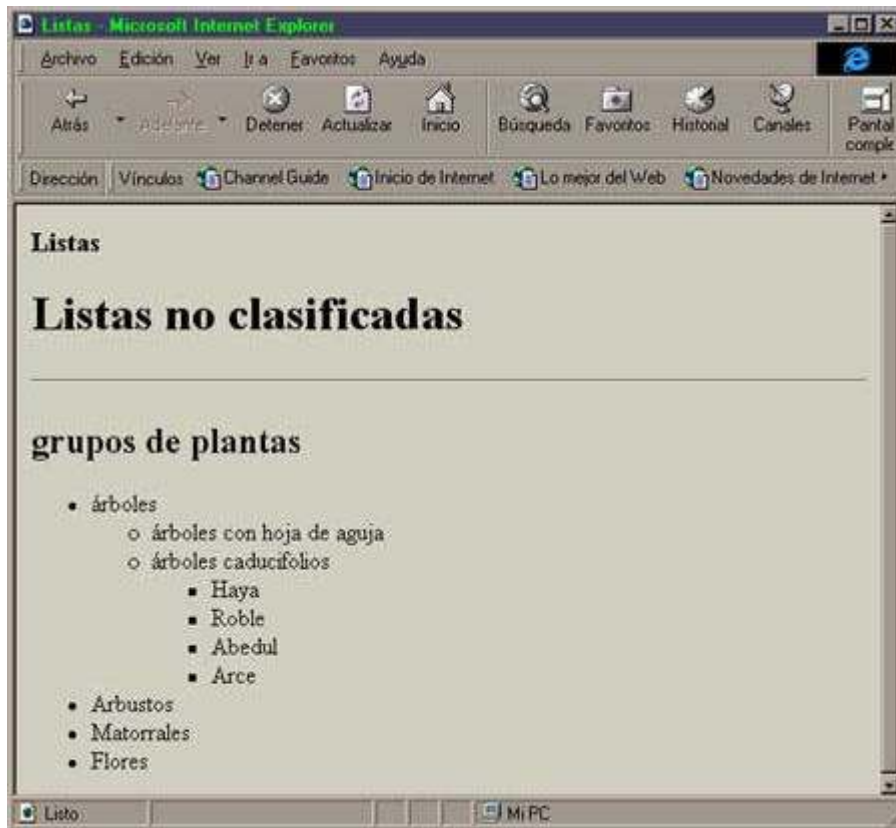
Tag	Significado
SAMP	Ejemplo
KBD	Introducción de datos por teclado
VAR	Variable
CITE	Cita
Q	Cita breve escrita entre comillas.
LANG	Texto breve en otra lengua.
AU	Nombre del autor.
DFN	Definición.
PERSON	Nombre de una persona; este tag sirve principalmente para crear directorios de personas.
ACRONYM	Acrónimo
ABBREV	Abreviatura
INS	Texto insertado
DEL	Texto borrado o a borrar.
SUB	Subíndice.
SUP	Superíndice.

3. Listas

Con HTML se pueden realizar diferentes tipos de listas. En primer lugar mencionaremos la lista sin clasificar, marcada con el elemento UL. Cada elemento de la lista va marcado con el elemento LI e HTML.

```
<HTML>
<HEAD>
<TITLE>Listas</TITLE>
</HEAD>
<BODY>
<H3>Listas</H3>
<H1>Listas no clasificadas</H1>
<HR>
<H2>grupos de plantas</H2>
<UL>
<LI>árboles</LI>
<UL>
<LI>árboles con hoja de aguja</LI>
<LI>árboles caducifolios</LI>
<UL>
<LI>Haya</LI>
<LI>Roble</LI>
<LI>Abedul</LI>
<LI>Arce</LI>
</UL>
</UL>
<LI> Arbustos</LI>
<LI>Matorrales</LI>
<LI>Flores</LI>
</UL>
</BODY>
</HTML>
```

Internet Explorer o Netscape Navigator agregan un punto, conocido como "bullet", delante de cada entrada de la lista.

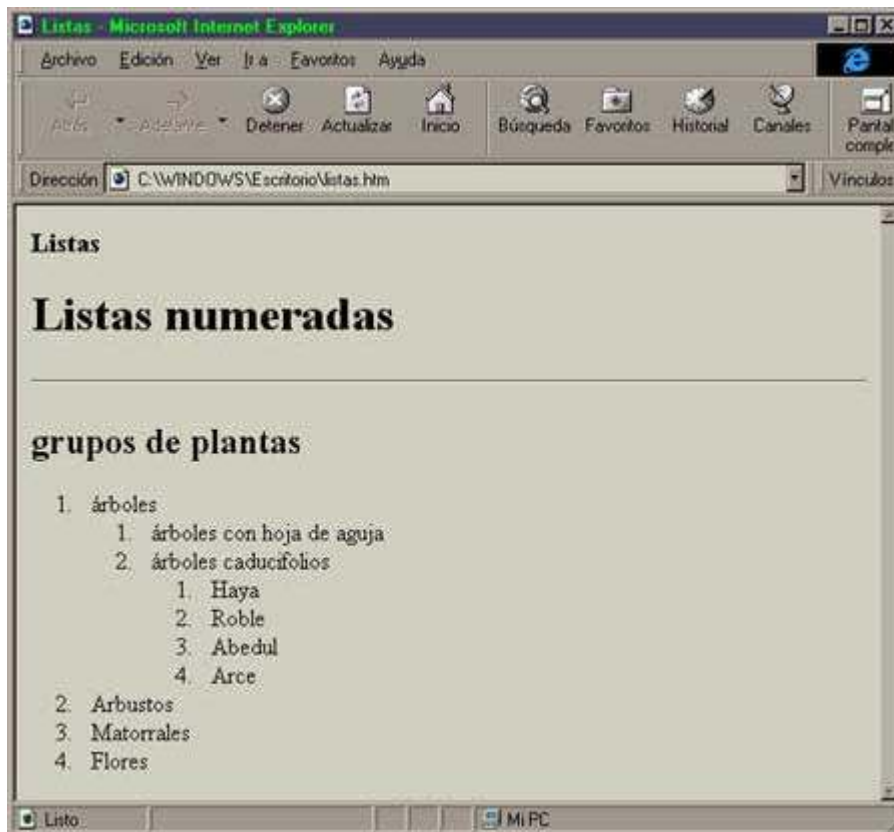


Las listas cuyos elementos se numeran son listas numeradas, "ordered lists", y van señalizadas por el elemento LO de HTML. Cada una de las entradas de la lista se forma, nuevamente, con el elemento LI (List Item).

```
<HTML>
<HEAD>
<TITLE>Listas</TITLE>
</HEAD>
<BODY>
<H3>Listas</H3>
<H1>Listas no clasificadas</H1>
<HR>
<H2>grupos de plantas</H2>
<OL>
<LI>árboles</LI>
<OL>
<LI>árboles con hoja de aguja</LI>
<LI>árboles caducifolios</LI>
<OL>
<LI>Haya</LI>
<LI>Roble</LI>
<LI>Abedul</LI>
<LI>Arce</LI>
</OL>
</OL>
<LI> Arbustos</LI>
<LI>Matorrales</LI>
```

```
<LI>Flores</LI>  
</OL>  
</BODY>  
</HTML>
```

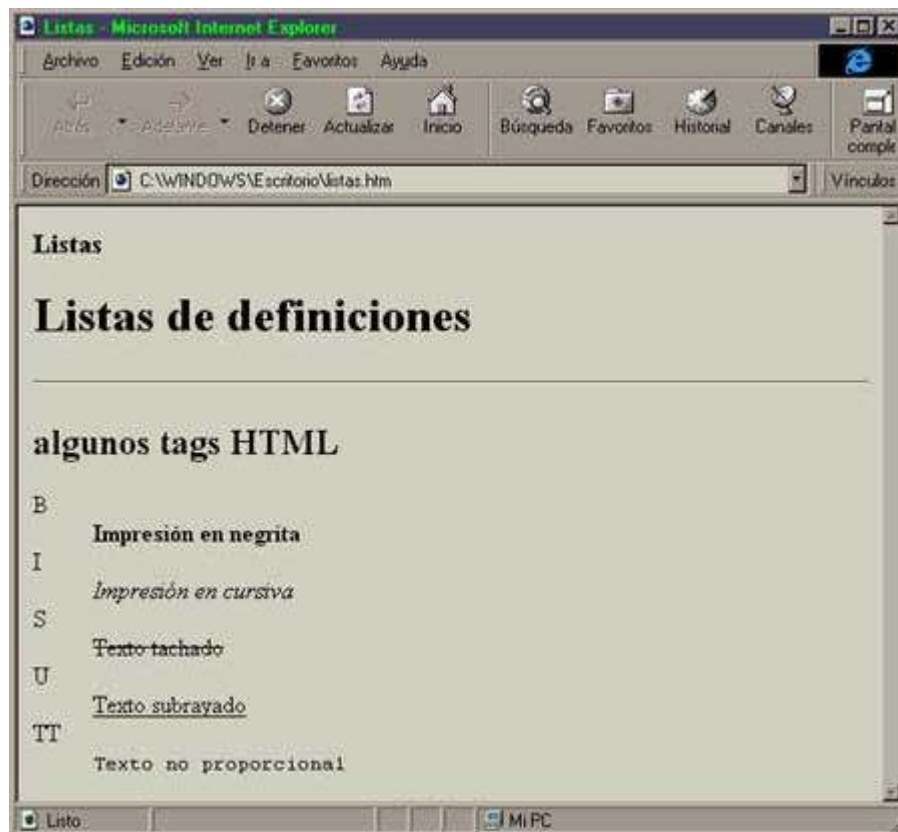
NS Navigator o MS Internet Explorer agrega el número correspondiente a cada entrada de la lista:



En las listas de definiciones creadas con el elemento DL cada entrada de lista consta de dos partes: el elemento DT, que contiene el concepto que debe ser definido o explicado, y el elemento DD que proporciona la descripción o la definición.

```
<HTML>  
<HEAD>  
<TITLE>Listas</TITLE>  
</HEAD>  
<BODY>  
<H3>Listas</H3>  
<H1>Listas de definiciones</H1>  
<HR>  
<H2>algunos tags HTML</H2>  
<DL>  
<DT>B</DT><DD><B>Impresión en negrita</B></DD>  
<DT>I</DT><DD><I>Impresión en cursiva</I></DD>  
<DT>S</DT><DD><S>Texto tachado</S></DD>  
<DT>U</DT><DD><U>Texto subrayado</U></DD>  
<DT>TT</DT><DD><TT>Texto no proporcional</TT></DD>  
</DL>  
</BODY>
```

</HTML>



4. Imágenes y enlaces

Una de las características fundamentales de HTML es el hecho de que se pueden vincular entre sí diferentes documentos y tipos de documentos. Las imágenes vinculadas al texto, designadas también como imágenes in-line, son ya un ejemplo de este tipo de vínculo. La vinculación se realiza mediante el tag de HTML. En el tag debe estar asignado el atributo SRC (Source, fuente) del URL del archivo de imágenes. Así un tag típico podría ser el siguiente:

```
<IMG SRC="http://www.nombrepagina.es/imágenes/imagen1.jpg">
```

Los hiperenlaces se realizan con el tag ancla <A>. El elemento HTML correspondiente es un contenedor, es decir, debe cerrarse con el tag final de forma . En el tag <A> se debe asignar al atributo HREF la dirección (que se activará pulsando el hiperenlace) en forma de un URL. En la siguiente línea HTML, la frase "Universidad de Murcia" representa un hiperenlace al ordenador de nuestra Universidad:

```
El laboratorio de redes se encuentra en el sótano de la Facultad de  
Informática de la <A HREF="http://www.um.es">Universidad de  
Murcia</A>.
```

También podemos colocar varios enlaces en una sola frase, constituyendo lo que se ha dado en llamar con uno de esos términos tan extraños que elegimos en este mundo de la informática, hipertexto.

El <A HREF=" <http://aries.dif.um.es>">laboratorio de redes se encuentra en el sótano de la Facultad de Informática de la <A HREF=" <http://www.um.es>">Universidad de Murcia.

A menudo se utilizan imágenes como hiperenlaces. Para ello es necesario anidar los tags y <A> descritos anteriormente.

Falta mencionar que el URL de un documento que se encuentra en el mismo ordenador y el mismo directorio que el documento que se quiere activar, se puede indicar en forma abreviada. En este caso es suficiente con indicar el nombre del archivo. El siguiente ejemplo ofrece la sintaxis de un hiperenlace con forma de imagen:

```
<A HREF="http://www.um.es">  
<IMG SRC="logounimurcia.jpg">  
</A>
```

Podríamos utilizar la opción de BORDER para hacer que el borde que aparece alrededor de una imagen con hiperenlace, desaparezca, dando una imagen mucho más adecuada a la calidad que requiere una buena web.

```
<A HREF="http://www.um.es">  
<IMG SRC="logounimurcia.jpg">  
</A>
```

5. Formularios

En los documentos HTML, mediante el tag <FORM>, se pueden crear formularios en los que el usuario puede introducir los datos de diversas maneras. La idea original de los formularios HTML prevé una evaluación del contenido del formulario por parte del servidor. Para ello en el atributo ACTION del tag <FORM> se indica el URL al que se deben enviar las informaciones introducidas en el formulario. Detrás del URL correspondiente se oculta un programa que se ejecuta en el servidor y que funciona conjuntamente con el programa servidor, por ejemplo, a través del puerto CGI; este programa es también el que crea la página web que se envía al browser como respuesta. En el elemento FORM se pueden crear los diferentes componentes del formulario con tres tags de HTML que presentamos en la siguiente lista:

Elemento del formulario	Significado
INPUT	Sirve para crear elemento de introducción de datos pequeños y diversos.
SELECT	Sirve para crear listas de selección.

Elemento del formulario	Significado
TEXTAREA	Sirve para definir un cuadro de texto de varias líneas.

El elemento más universal aquí es el elemento INPUT que puede tomar diferentes formas según sea el valor del atributo TYPE (atributo obligatorio). En la tabla siguiente se muestran los valores posibles para el atributo TYPE.

TYPE=	Significado
TEXT	Cuadro de texto de una sola línea.
PASSWORD	Cuadro de texto para contraseñas. Los caracteres escritos no se representan en el browser.
CHECKBOX	Casilla de verificación.
RADIO	Botón radio.
SUBMIT	Botón estándar que sirve para enviar el contenido del formulario al servidor.
RESET	Botón estándar. Sirve para anular el contenido del formulario; es decir, todos los contenidos vuelven a ser los predeterminados.
IMAGE	Imagen.
HIDDEN	Elemento oculto.
BUTTON	Botón en general cuya función debe definirse mediante un programa script.

Todos los elementos INPUT poseen los atributos NAME y VALUE. Finalmente, al servidor se le transmiten pares de valores resultantes de las informaciones asignadas a estos atributos. El valor del atributo VALUE se define a partir de las introducciones realizadas por el usuario.

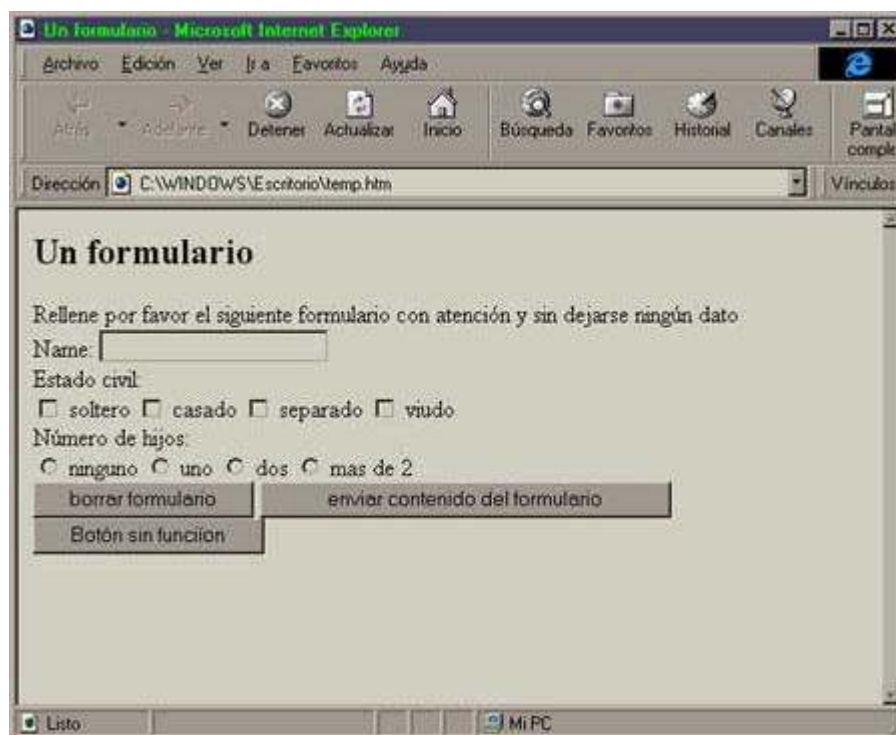
Dentro de un elemento SELECT suele haber, por lo general, varios tags <OPTION> que son los que definen cada una de las líneas de la lista de selección.

El siguiente ejemplo presenta un formulario HTML que contiene prácticamente todos los tipos de elementos disponibles:

```
<HTML>
<HEAD>
<TITLE>Un formulario</TITLE>
</HEAD>
<BODY>
<H2>Un formulario</H2>
<FORM ACTION="?">
Rellene por favor el siguiente formulario con atención y sin dejarse
ningún dato<BR>
Name:
<INPUT TYPE="text" NAME="nombre">
<contraseña:
<INPUT TYPE="password" NAME="pass">
```

```
<BR>
Estado civil:<BR>
<INPUT TYPE="CHECKBOX" NAME="soltero" VALUE="0"> soltero
<INPUT TYPE="CHECKBOX" NAME="soltero" VALUE="0"> casado
<INPUT TYPE="CHECKBOX" NAME="soltero" VALUE="0"> separado
<INPUT TYPE="CHECKBOX" NAME="soltero" VALUE="0"> viudo<BR>
Número de hijos:<BR>
<INPUT TYPE="RADIO" NAME="hijo" VALUE="0"> ninguno
<INPUT TYPE="RADIO" NAME="hijo" VALUE="1"> uno
<INPUT TYPE="RADIO" NAME="hijo" VALUE="2"> dos
<INPUT TYPE="RADIO" NAME="hijo" VALUE="3"> mas de 2<BR>
<INPUT TYPE="RESET" VALUE="borrar formulario">
<INPUT TYPE="SUBMIT" VALUE="enviar contenido del formulario">
<INPUT TYPE="button" NAME="botón" VALUE="Botón sin funcion">
</FORM>
</BODY>
</HTML>
```

Resulta especialmente remarcable el hecho de que todos los elementos del tipo RADIO tienen el mismo nombre. Como consecuencia, en el formulario representado por el browser correspondiente sólo se puede seleccionar una de las opciones dotadas con estos botones. No sucede lo mismo al utilizar el tipo CHECKBOX, pues con él se puede seleccionar en el formulario que uno está casado y soltero a la vez.



6. Tablas

La sintaxis de las tablas ampliadas, comparativamente compleja, es compatible hacia delante con las tablas HTML usuales, definidas por primera vez en la versión HTML 3.0 y que presentan la misma forma

que las de la versión HTML 3.2. En la tabla siguiente ofrecemos resumida la estructura de esta sintaxis de tablas tan extendida:

Elemento	Significado	Elementos que contiene
TABLE	La tabla completa	CAPTION (opcional); TR (una o más veces)
CAPTION	Título	-
TR	Fila de tabla	TH y TD (en el orden y la frecuencia que se desee)
TH	Campo de título	-
TD	Campo de datos	-

El resumen siguiente permite apreciar la estructura más compleja de la nueva sintaxis ampliada:

Elemento	Significado	Elementos que contiene
TABLE	La tabla completa	CAPTION (opcional) COL o COLGROUP (tantas veces como se desee) THEAD (opcional) TFOOT (opcional) TBODY (una o más veces)
CAPTION	Título	-
THEAD	Área de cabecera de la tabla	TR (una o más veces)
TFOOT	Área de pie de la tabla.	TR (una o más veces)
TBODY	Cuerpo de la tabla.	TR (una o más veces)
COLGROUP	Grupo de definición de columnas	COL (las veces que se desee)
COL	Definición de columnas	-
TR	Fila de tabla	TH y TD (en el orden y frecuencia que se desee)
TH	Campo de título	-
TD	Campo de datos	-

Hay que señalar que la compatibilidad hacia delante entre ambas estructuras de tabla no es evidente. Sin embargo, se desprende del hecho de que el elemento de tabla **TBODY** no tiene que ir marcado explícitamente con los tags **<TBODY>** y **</TBODY>**. Si faltan los elementos **THEAD** y **TFOOT**, los elementos **TR** que se encuentran directamente en el elemento **TABLE** son considerados implícitamente como elementos del cuerpo de la tabla. Y eso se corresponde con la

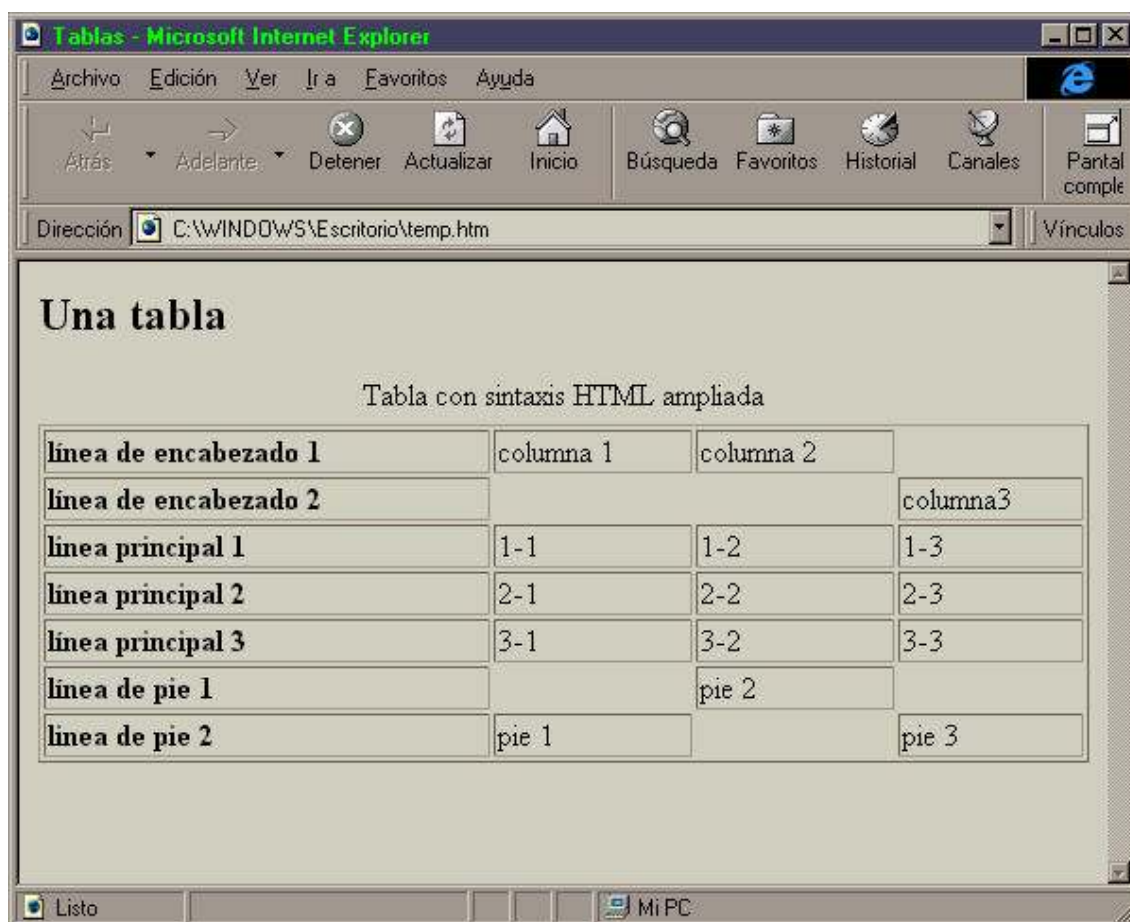
evaluación de las tablas HTML 3.0. Los tags finales de las tres áreas de la tabla pueden omitirse. En el documento HTML del ejemplo siguiente se ha utilizado la nueva sintaxis:

```
<HTML>

<HEAD>
<TITLE>Tablas</TITLE>
</HEAD>
<BODY>
<H2>Una tabla</H2>
<TABLE BORDER WIDTH="100%">
<CAPTION>Tabla con sintaxis HTML
ampliada</CAPTION>
<COL SPAN="4" ALIGN="left">
<THEAD>
<TR>
<TH>línea de encabezado 1
<TD>columna 1<TD>columna 2<TD>
<TR>
<TH>línea de encabezado 2
<TD><TD><TD>columna3
<TFooter>
</BODY>

<TR>
<TH>línea de pie 1
<TD><TD>pie 2<TD>
<TR>
<TH>línea de pie 2
<TD>pie 1<TD><TD>pie 3
<TBODY>
<TR>
<TH>línea principal 1
<TD>1-1<TD>1-2<TD>1-3
<TR>
<TH>línea principal 2
<TD>2-1<TD>2-2<TD>2-3
<TR>
<TH>línea principal 3
<TD>3-1<TD>3-2<TD>3-3
</TABLE>

</HTML>
```



Sin embargo, la presentación que hacen los browsers en versiones inferiores a las 4.0 no son satisfactorias, pues el área de pie de la tabla queda sobrepuesta al cuerpo de la tabla.

7. Mapas de imágenes

Por mapa de imagen se entiende cualquier imagen que contenga varios hiperenlaces. La imagen en conjunto no es un hiperenlace sino que contiene diversas áreas, denominadas zonas hot, con diferentes enlaces asignados a cada una de ellas. La idea original de la WWW contaba con que al pulsar con el ratón en un mapa de imágenes el servidor evaluaría esta acción. Pero pronto se vio que esto provocaría una circulación innecesaria de datos y la consiguiente saturación del servidor. Consecuentemente, se desarrolló un nuevo concepto de mapas de imágenes que pudieran ser evaluados por el cliente. Con estos nuevos mapas de imágenes era el browser quien se encargaba de asignar el hiperenlace correspondiente cada vez que se pulsaba una imagen con el ratón.

Este concepto propuesto dentro del marco de HTML 3.0 sólo fue implantado en algunos browsers y ha quedado en la actualidad. En HTML 3.2 se trabaja con un nuevo concepto que es soportado también por IE 3.0, Netscape 4.0 e IE 4.0. En la tabla siguiente se muestra la estructura sintáctica de este concepto:

Elemento	Significado	Uso
IMG	Sirve para reproducir la imagen que se quiere utilizar como mapa de imagen.	A través del atributo USEMAP se crea la conexión con el elemento MAP correspondiente. Al nombre se le debe anteponer al carácter "#".
MAP	Elemento marco para establecer la correspondencia entre zonas hot y enlaces.	El atributo obligatorio NAME sirve para identificar el elemento MAP. MAP uno de los elementos AREA.
AREA	Establece una correspondencia entre una zona hot y un URL.	Contiene al menos los atributos SHAPE, COORD y HREF. Otros atributos posibles son NOHREF y ALT.

El punto central de la sintaxis del mapa de imágenes en la parte del cliente es, por lo tanto, el tag <AREA>. A través de los atributos de este tag se controlan las diferentes funciones. El atributo SHAPE indica la forma de la zona hot correspondiente; y los valores posibles aquí son los atributos RECT, CIRCLE y POLYGON que indican, respectivamente, si se trata de una zona hot rectangular, circular o poligonal. El atributo COORDS contiene todas las coordenadas necesarias en forma de variable de carácter. Esta variable consta de números enteros separados entre sí por una coma. En el caso de SHAPE=RECT éstos son el par de valores que indican la posición de los vértices superior

izquierdo e inferior derecho del rectángulo. Cuando SHAPE=CIRCLE son necesarios tres números para indicar las coordenadas del centro del círculo y el radio del mismo. Si SHAPE=POLYGON la lista de valores consta entonces de pares de valores cada uno de los cuales indica la posición de un vértice del polígono.

Finalmente, el atributo HREF indica la dirección del documento que puede abrirse pulsando simplemente en la zona correspondiente. Con el atributo ALT se puede predefinir un texto alternativo que podrá reproducirse en lugar de la imagen cuando se trabaje con un browser no gráfico.

Naturalmente, también puede suceder que, dentro de un mapa de imágenes, haya algunos puntos que tengan asignadas varias zonas "hot" a la vez, pues no se puede excluir la posibilidad de que las áreas geométricas definidas con AREA queden parcialmente superpuestas. Cuando esto sucede, el hiperenlace que actúa siempre es aquel cuyo elemento AREA aparece dentro del elemento MAP. Este hecho hace posible y conveniente en general la asignación de un último elemento AREA que defina toda la imagen como zona hot y le asigne la dirección que debe ser activada cuando al pulsar con el ratón no se alcanza ninguna de las restantes zonas hot.

En el ejemplo siguiente se han elegido como destinos de los hiperenlaces nombres inventados que indican el color y la forma de la zona hot correspondiente en cada caso:

```
<HTML>
<HEAD>
<TITLE>Mapa de imágenes en el cliente</TITLE>
</HEAD>
<BODY>
<H2>Mapa de imágenes en el cliente</TITLE>
<IMG SRC="csim.jpg" USEMAP="#hotzones">
<MAP NAME="hotzones">
<AREA SHAPE="polygon" HREF="http://rotes.polygon"
COORDS="0,250,100,150,200,150,350,250">
<AREA SHAPE="polygon" HREF="http://grunes.polygon"
COORDS="50,70,250,75,125,200,50,125">
<AREA SHAPE="RECT" HREF="http://blaudes.quadral"
COORDS="0,0,100,100">
<AREA SHAPE="CIRCLE" HREF="http://gelber.circular" COORDS="250,75,75">
<AREA SHAPE="RECT" HREF="http://weiber.hintergrund"
COORDS="0,0,350,250">
</MAP>
</BODY>
</HTML>
```

mientras el mapa de imágenes se representa en el browser, el contacto del puntero del ratón con las zonas hot se registra y las direcciones del salto se emiten en la barra de estado, tal como sucede con los hiperenlaces normales.

8. Frames (marcos)

El concepto de frame, es decir, de la división interna de páginas web en ventanas parciales, fue introducir por la casa Netscape en la versión 2.0 de Netscape Navigator. Este concepto está en clara contradicción con las especificaciones oficiales de HTML, en las que, a pesar del esfuerzo realizado, no se ha podido incorporar todavía. De cualquier modo, el concepto de frame fue adoptado rápidamente, poco después de su presentación, por muchos diseñadores de páginas web y en la actualidad su uso es tan frecuente y variado que es prácticamente imposible pensar en la WWW sin él.

La idea fundamental de los frames y su conversión sintáctica son relativamente sencillas. El objetivo principal era estructurar la ventana principal del browser en varias ventanas parciales en las que presentar diferentes documentos HTML. Una página de la web que utilice frames consta por eso mismo de varios archivos. Hay un archivo o varios que definen el tamaño y el orden de los frames y, además, los archivos que representan las páginas HTML. Es decir, hay una división de forma y contenido. En los archivos HTML "de forma" se utilizan básicamente los dos nuevos tags `<FRAMESET>` y `<FRAME>`.

Con el atributo COLS se define cuántas ventanas parciales del FRAMESET correspondiente se han de ordenar alineadas y qué anchura deben tener las mismas. El valor necesario de este atributo es una variable de caracteres formada por números enteros separados entre sí por una coma; estos números representan la anchura en puntos de pantalla. Uno de estos números se puede reemplazar por el carácter "*", y en ese caso será el browser quien defina la anchura de la columna correspondiente. Cuando los números van seguidos del carácter de tanto por ciento, esto significa que los números no se interpretan como puntos de pantalla, sino como porcentaje respecto al ancho de la ventana del browser. El tag `<FRAMESET COLS="30%,30%,40%">` divide la pantalla del browser en tres ventanas parciales, las dos primeras de las cuales ocupan un 30% de la anchura total y la tercera el 40% restante.

El atributo ROWS divide la ventana del browser en ventanas parciales horizontales. La sintaxis en este caso es idéntica a la del atributo COLS. El tag `<FRAMESET ROWS="100,*">`, por ejemplo, tiene por efecto la división de la ventana del browser en una ventana superior de 100 puntos de pantalla de altura y otra subventana inferior que ocupa el resto de la superficie libre.

Explicaremos a continuación el tag `<FRAME>`. En un elemento FRAMESET hay tantos `<FRAME>` como ventanas parciales resulten de los atributos COLS y ROWS. El dato principal de un tag `<FRAME>` es el atributo SRC, que contiene el URL del documento que debe mostrarse en la ventana parcial correspondiente. Con el atributo NAME se puede

asignar a la ventana parcial un nombre que podrá utilizarse para acceder y referirse a la ventana sin equívocos.

Con los atributos `MARGINWIDTH` y `MARGINHEIGHT` se pueden definir las medidas y los márgenes de las ventanas parciales. El atributo `SCROLLING`, que puede adoptar los valores "yes", "no" y "auto", se define si la ventana parcial debe tener barras de desplazamiento o no. El valor "yes" hace que las barras de desplazamiento se representen siempre en la ventana; el valor "no" hace que no se representen nunca y el valor "auto" deja que sea el browser quien decida automáticamente, según el tamaño de la ventana, si debe incluir en ésta barras de desplazamiento o no. Por último, con el atributo `NORESIZE`, al que no se le asigna ningún valor, se puede definir que el tamaño de la ventana permanezca siempre constante independientemente de los cambios eventuales del tamaño de la ventana del browser.

Los browsers que no pueden evaluar los tags `<FRAMESET>` y `<FRAME>` no pueden representar ningún documento en cuya creación se hayan utilizado estos tags. Por eso, para dejar abierta la posibilidad de mostrar texto HTML normal como alternativa al contenido de los `FRAME`, por ejemplo para sugerir la conveniencia de trabajar con otro browser, se ha introducido el tag `<NOFRAMES>`. El elemento `NOFRAMES` debe poseer siempre tanto tag inicial como final y el texto entre ambos se evalúa como el de un documento HTML normal.

Como ejemplo ilustrativo sobre el uso de los comandos descritos se reproduce a continuación el listado del archivo HTML. Este archivo define la división en frames de una web:

```
<HTML>
<HEAD>
<TITLE>Ventana principal</TITLE>
</HEAD>
<FRAMESET ROWS="100,*">
  <FRAME SRC="menu.htm" NAME="menu">
  <FRAMESET COLS="140,*">
    <FRAME SRC="publi.htm" NAME="publi">
    <FRAME SRC="main.htm" NAME="principal">
  </FRAMESET>
</FRAMESET>
</HTML>
```

A los frames definidos aquí se les ha asignado un nombre a través del atributo `NAME`, ¿Por qué era necesario?

Quando se pulsa con el ratón un hiperenlace normal de un documento representado en un frame, el documento al que hace referencia el enlace se carga inmediatamente y se muestra en el mismo frame. Claro que esto no es siempre lo más conveniente. Es más, muchas veces se encontrará con páginas web con un frame que hace las funciones, por así decir, de menú; pero los documentos que se activan desde esa página se muestran en otro frame. Por lo tanto debe ser posible seleccionar uno de los frames existentes como destino para un hiperenlace. Esto es lo que permite seleccionar uno de los frames

existentes como destino para un hiperenlace. Esto es lo que permite precisamente el atributo TARGET del tag ancla <A>. Como ejemplo ilustrativo reproducimos a continuación el principio de un archivo:

```
<HTML>
<HEAD>
<TITLE>Columna de contenido</TITLE>
<HEAD>
<BODY>
<H1>Menu</H1>
<A HREF="1.htm" TARGET="main">opcion 1 </A>
<BR>
...
```

Como podrás ver, todos los documentos a los que se puede acceder por hiperenlaces pueden ser desviados a la mayor de las ventanas parciales asignándole al atributo TARGET el valor "main". Si no se le asignara este valor al atributo, los documentos se presentarían en estrechas columnas situadas en la parte izquierda de la pantalla, lo que, naturalmente, resultaría incómodo y un tanto absurdo.

El atributo TARGET puede adoptar, además de los nombres de frame, determinados valores estándar que presentamos en la tabla siguiente:

Valor	Significado
"_blank"	Carga el documento en una nueva ventana del browser.
"_self"	Carga el documento en la misma ventana desde donde se llama.
"_parent"	Carga el documento en la ventana inmediatamente superior dentro de la jerarquía de FRAME.
"_top"	Carga el documento en la ventana superior de la jerarquía de FRAME.

9. Meta elementos

El elemento <META> se usa dentro del elemento <HEAD> para introducir información que no está definida en otros elementos HTML. Esta información puede ser extraída por clientes o servidores para identificar, indexar y catalogar documentos especializados.

A pesar de que se trate de información general, es preferible usar elementos con nombres particulares como el título.

Los servidores HTTP pueden leer la información de las cabeceras correspondientes a cualquier elemento definiendo un valor para el atributo HTTP-EQUIV. Esto da al autor del documento un mecanismo (que no tiene porqué ser el mejor) para identificar la información que debe ser introducida en las cabeceras de respuesta en una petición HTTP.

Los atributos de META los enumeramos a continuación.

CONTENT

El contenido de la metainformación se asocia con la información contenida en el documento.

Si el documento contiene:

```
<META HTTP-EQUIV="Expires" CONTENT="Sat, 06 Jan 1990 00:00:01 GMT">  
<META HTTP-EQUIV="From" CONTENT="nick@htm.com">  
<META HTTP-EQUIV="Reply-to" CONTENT="stephen@htm.com">
```

La cabecera de respuesta HTML sería:

```
Expires: Sat, Jan 1990 00:00:01 GMT  
From: nick@htm.com  
Reply-to: stephen@htm.com
```

Normalmente, los documentos HTML pueden contener muchos términos repetidos. Algunos motores de búsqueda usan la información sobre palabras clave generada por el servidor en forma de <META HTTP-EQUIV="Keywords" CONTENT="..."> para determinar el contenido del documento especificado y para calcular su nivel de relevancia.

No uses el elemento <META> para definir información que debe estar asociada con un elemento HTML existente. Por ejemplo, no sería lógico definir dos veces el elemento TITLE.

El elemento <META> es particularmente útil para construir documentos dinámicos. Usamos la siguiente sintaxis:

```
<META HTTP-EQUIV="Refresh" CONTENT="x">
```

Esto hace que el browser crea que el servidor ha indicado en la cabecera:

```
Refresh: x
```

Esto hace que el documento se actualize cada x segundos.

Esta utilidad tiene sentido cuando queremos redirigir al usuario a otra página para lo que utilizaríamos el elemento de la siguiente manera:

```
<META HTTP-EQUIV="Refresh" CONTENT="2;URL=http://www.pepe.com">
```

Para que nos hagamos idea, el sistema cargará la página actual, y dos segundos después, cargará www.pepe.com aunque no debemos olvidar colocar la dirección URL en formato completo para que no puedan haber confusiones.

11. El futuro de HTML

La definición de las especificaciones estándar del lenguaje HTML de la WWW es tarea del World Wide Web Consortium ("W3C"). Los comandos definidos en su día para la creación de fórmulas matemáticas así como algunos otros componentes de su especificación, no fueron implementados por las grandes casas productoras de browsers. Por otra parte, los comandos necesarios para la vinculación de contenidos animados y activos dentro de la Web fueron implementados por Netscape y Microsoft antes de que entraran a formar parte de las especificaciones oficiales. La tabla siguiente muestra un resumen de los tags HTML más relevantes en este contexto.

Tag	Explicación
SCRIPT	Sirve para vincular programas script. Introducido por Netscape (Navigator 2.0). Componente reciente de la especificación HTML 3.2. Implementado en Internet Explorer 3.0
APP	Sirve para vincular applets de Java. Introducido por Sun Microsystems. No es componente oficial de HTML. Actualmente anticuado.
APPLET	Sirve para vincular applets de Java. Forma parte de la especificación HTML 3.2. Implementado en Internet Explorer 3.0.
EMBED	Sirve para vincular elementos de programas en general (plugins en Netscape).Introducido por Netscape (Navigator 2.0). No es componente oficial de HTML. Implementado en Internet Explorer 3.0
INSERT	Sirve para vincular objetos de programas en general. Introducido por Microsoft. No es componente oficial de HTML. Actualmente anticuado. Antecesor de OBJECT.
PARAM	Sirve para la transmisión de parámetros dentro de elementos de APPLET y OBJECT. Componente de HTML 3.2. Implementado en Internet Explorer 3.0 y Netscape Navigator 3.0.

Con los tags mencionados se pueden crear puertos de comunicación entre HTML y diversos lenguajes de programación y tecnologías que amplían considerablemente las posibilidades de la WWW. Por eso el diseñador de páginas web del futuro deberá dominar, junto a HTML, todas las técnicas que se ocultan tras los mencionados comandos. No obstante, HTML es y sigue siendo base fundamental e imprescindible para cualquier tipo de oferta de información en la WWW.

C. Gráficos

1. ¿Qué son los gráficos para web?
 - 1.1. Iconos
 - 1.2. Fotografías e ilustraciones
 - 1.3. Gráficos de fondo (tapices)
2. Los diferentes tipos de imágenes
 - 2.1. GIF (Graphic Interchange Format)
 - 2.2. JPEG (Joint Photographic Experts Group)
3. Tags HTML básicos para añadir imágenes
4. Tamaño de los pixels
5. Elegir un número correcto de colores
6. Formatos GIF
7. Formatos JPEG
8. Resumen de utilización de Paint Shop Pro.
9. Filtros para imágenes
10. Deformaciones para imágenes
11. Combinar y solapar las imágenes
12. Efectos de color
 - 12.1. Grey Scale y Colorize
 - 12.2. Negative y Solarize
 - 12.3. Posterize y Decrease Color Depth
13. Efectos especiales
 - 13.1. Drop Shadows y Highlights
 - 13.2. Crear botones 3D
 - 13.3. Hot Wax y Tinting
14. Hacer gráficos eficientes
 - 14.1. ¿Porqué utilizar gráficos eficientes?

- 14.2. Recorte, redimensionado y reducción de imágenes.
 - 14.2.1. Redimensionar una imagen.
 - 14.2.2. Haciendo thumbnails (reducciones)
 - 14.2.3. Recortar
- 15. ¿Cuántos colores están bien para una imagen GIF?
 - 15.1. Cómo afectan los colores al tamaño de un fichero GIF
 - 15.2. Reducir los colores
- 16. Compresión JPEG
- 17. Usar Interlaced y Progressive
- 18. Gif animados
 - 18.1. Barra de herramientas
 - 18.2. Pestaña Opciones
 - 18.3. Pestaña animación
 - 18.4. Pestaña Imagen
- 19. Recursos sobre gráficos en Internet

C. Gráficos

1. ¿Qué son los gráficos para web?

Simplemente diremos que los gráficos para web son imágenes grabadas en el ordenador que han sido creadas y optimizadas específicamente para una página web. Trabajando en conjunción con tags especiales de HTML, los gráficos pueden añadirse en cualquier página web en cualquier momento. Por supuesto, tienen que ser grabados en un formato determinado que pueda ser interpretado por los browsers de web más populares.

Ahora pasamos a ver los principales tipos de gráficos que nos encontraremos al explorar la web:

1.1. Iconos

El tipo más común de gráficos que te encontrarás son los iconos. Los iconos son gráficos pequeños que se usan para representar gráficamente otros comandos o acciones. En la WWW, los iconos normalmente se enlazan con otras páginas web en lugar de mediante texto normal, sirven como una forma gráfica de conectar página unas con otras. En general, los iconos desempeñan un doble papel: mejoran la apariencia de un sitio web y guían a los visitantes de una página a otra.

1.2. Fotografías e ilustraciones

La mayoría de las páginas web personales normalmente tienen algún tipo de fotografías o ilustraciones. Para poner estas y fotografías similares en una página web, tienes que escanearlas en el ordenador y añadir el HTML a su página web.

Además de añadir fotografías a las páginas web, algunos sitios web también incluyen logos creados o ilustraciones que reflejan el tema en que se basa el sitio.

1.3. Gráficos de fondo (tapices)

Otra forma normal de usar los gráficos en una página web es añadirlos al fondo, debajo del texto e información que normalmente aparece en su sitio web. El efecto es como colocar las letras sobre un plato con unos dibujos: puedes ver las letras cubriendo los tapices.

Los gráficos de fondo son herramientas normales para añadir un poco de vida a las páginas web. Puedes crear y usar una variedad de tapices para aplicar diferentes diseños, colores y estilos a tu sitio web. Es importante señalar que los fondos añaden texturas a las simples pantallas blancas, además de añadir algún tipo de personalidad a su sitio web.

2. Los diferentes tipos de imágenes

Como puedes imaginar, todas las imágenes no tienen porqué estar salvadas en el mismo formato. Así como hay diferentes tipos de gráficos que puedes usar en tu página web, también hay diferentes tipos de formatos que puedes utilizar para grabarlas. La mayoría de browsers de internet, pese a todo, sólo reconocen dos formatos gráficos estándar: GIF y JPEG.

A pesar de que hay muchos estándares, la principal diferencia entre ellos se centra en la compresión de los ficheros. Cuando un gráfico se salva en el ordenador, se transforma en un montón de ceros y unos.

Algunos formatos gráficos simplemente escriben todos los ceros y unos en un fichero enorme. Esto da como resultado ficheros muy grandes, pero el ordenador no tiene que calcular demasiado para convertir los binarios en un gráfico visualizable. Desafortunadamente, las imágenes de este tipo tienden a tener un tamaño mucho más grande, normalmente de cientos de bytes en el ordenador para una imagen que parece casi ridícula en la pantalla del ordenador.

Otros formatos usan diferentes métodos para grabar números binarios, que dan como resultado ficheros más pequeños. Usando técnicas de compresión, tu ordenador reemplaza unas series repetitivas de números binarios con un número más pequeño, resultando un fichero más pequeño. Cuando trabajamos con este tipo de ficheros, el ordenador tiende a tomar unos segundos extra para descomprimir las imágenes antes de mostrarlas en pantalla.

La mayoría de los diferentes formatos son combinaciones de rendimiento, tamaño de fichero y técnicas de compresión. Es importante, tener presente que un gráfico puede ser grabado en diferentes formatos gráficos y cada formato tendrá un tamaño diferente, y una calidad de fotografía.

2.1. GIF (Graphic Interchange Format)

Este tipo de fichero fue creado por Compuserve para proporcionar información en un formato gráfico estándar. GIF es un estándar desde

hace aproximadamente diez años y fue el primero soportado por la WWW.

El formato de imágenes GIF usa un popular algoritmo de compresión denominado Lempel-Ziv-Welsh, la forma más fácil y eficiente de comprimir ficheros en el menor espacio posible. Además, es el formato más popular en todo el mundo. En la compresión y en la descompresión, el fichero no pierde ningún detalle, todos los colores se mantienen y no cambia la apariencia de las imágenes.

Las imágenes GIF sólo pueden tener un máximo de 256 colores diferentes en un fichero.

2.2. JPEG (Joint Photographic Experts Group)

También es conocido vulgarmente como JPG, este formato fue desarrollado para ser significativamente más eficiente que los GIFs en determinadas circunstancias, especialmente en las imágenes grandes con muchos colores. JPEG utiliza un algoritmo de compresión más avanzado que el GIF, y este algoritmo convierte los gráficos en más pequeños.

El algoritmo de compresión de JPEG trabaja de forma diferente al formato GIF, pero también tiene algunas pérdidas. Los JPEGs usan un algoritmo que pierde algo de detalle cuando grabamos y vemos las fotografías en este formato. Como resultado los ficheros JPG no son tan detallados como las imágenes GIF, pero pueden ofrecer hasta un 35% de mejora en el tamaño del fichero y compresión.

No debemos preocuparnos sobre la pérdida de detalle en la compresión. Cuando grabamos los JPEG, tenemos la opción de elegir cuando detalle queremos perder. A mayor detalle, mayor tamaño de fichero, de forma que al llegar al mayor detalle, los ficheros ocupan casi lo mismo que los GIF.

Además, ya que los JPEG fueron desarrollados para manejar fotografías, son mucho más eficientes a la hora de manejar muchos colores y formas diferentes. Esto significa, que los JPEG tienden a ser más pequeños y por tanto, se descargan más rápido al explorar las webs. Esto hace las imágenes JPEG más atractivas para los desarrolladores de web porque los visitantes pueden ver sus imágenes mucho más rápido. El formato JPG soporta hasta 16.7 millones de colores diferentes, significativamente más que los ficheros GIF.

La mayoría de los browsers soportan tanto los JPG como los GIF.

3. Tags HTML básicos para añadir imágenes

El tag para añadir imágenes a la web es el IMG. El tag IMG tiene como atributos SRC, ALIGN, ALT, VSPACE, HSPACE y BORDER.

```
<IMG SRC="avion.jpg">
```

Este tag nos permite situar un fichero llamado avion.jpg en nuestra web.

Ahora veremos como se usan el resto de tags con un ejemplo:

```
<IMG SRC="avion.jpg" ALIGN=RIGHT ALT="Gráfico de un avión">
```

Este tag insertará el mismo gráfico a la derecha (permitiendo que el texto que le siga se inserte a la izquierda de la imagen).

El atributo ALT nos indica el "texto alternativo" que aparecerá en lugar del gráfico cuando carguemos la web sin gráficos o cuando situemos el cursor encima del gráfico (en Internet Explorer 3.0 y superior y en Netscape Navigator 4.0).

```
<A HREF="aviones.htm">  
<IMG SRC="avion.jpg" ALIGN=RIGHT ALT="Gráfico de un avión">  
</A>
```

Como sabemos, al situar un enlace con un gráfico utilizando el código anterior, nos aparecerá un borde del mismo color que los enlaces del resto de la página. Para evitar esto, podemos utilizar el atributo BORDER del tag IMG, poniéndolo a cero, con lo que desaparecerá; si por el contrario queremos que el borde sea mayor, podemos aumentar su valor.

```
<A HREF="aviones.htm">  
<IMG SRC="avion.jpg" ALIGN=RIGHT ALT="Gráfico de un avión" BORDER=0>  
</A>
```

Además, si el ALIGN lo ponemos a LEFT, veremos que el texto que le siga al gráfico y el mismo gráfico aparecen completamente juntos, lo que puede confundir al usuario o hacer que la página no quede como deseamos. Para separar un gráfico del resto de elementos de la página por un borde invisible, utilizamos los tags VSPACE y HSPACE que nos permiten seleccionar el número de pixels que habrá de espacio Vertical u Horizontal.

```
<A HREF="aviones.htm">  
<IMG SRC="avion.jpg" ALIGN=LEFT ALT="Gráfico de un avión" BORDER=0  
VSPACE=10 HSPACE=10>  
</A>
```

4. Tamaño de los pixels

Antes de que podamos crear nuestra primera imagen, tenemos que decidir cómo de grande queremos que aparezca en la pantalla del ordenador. Decidir la altura y anchura de la imagen es extremadamente importante porque afecta a cómo veremos la imagen en el browser y tiene relación directa con el tamaño del fichero. Por tanto, el tamaño de la imagen afectará al tiempo que tarde la imagen en descargarse y mostrarse en una página web, una medida importante en el uso de la WWW. En general, queremos que la imagen sea lo más pequeña posible para que pueda verse inmediatamente.

La altura y anchura de tu pantalla está medida en pixels (picture elements). Un monitor estándar VGA puede mostrar 640 pixels de anchura y 480 de altura. Super VGA (SVGA) ofrece 800 por 600, y SVGA Mejorado ofrece 1024 por 768. Los pixels son pequeños puntitos que forman las imágenes que vemos en la pantalla.

Hablaremos principalmente sobre las características de resolución para ordenadores compatibles PC. Recuerda que la WWW da acceso universal a la información y todos los tipos de ordenadores tendrán acceso a la información e imágenes que coloquemos en nuestras páginas web. Algunos Macintosh y estaciones de trabajo de alto rendimiento Sun o Hewlett-Packard pueden tener resoluciones superiores. En general, podemos tomar como modelo los tamaños de pantalla de PC que puede ser nuestra guía para crear las páginas web.

Como regla general, desarrollaremos nuestras páginas web para la resolución de pantalla menor. Esto nos asegura que nuestras imágenes se podrán ver por cualquiera que navegue en la WWW sin problemas extra. Esto significa que debemos poner nuestros browsers aproximadamente a un tamaño de 640x480 a la hora de hacer las pruebas de visualización. De hecho, para ver correctamente las imágenes en todos los programas ninguna imagen debe tener más de 600 pixels de ancho y 440 de alto. Quitamos 40 pixels para cada eje de forma que nos aseguramos de que los gráficos caben dentro de los bordes de Netscape o IE.

Una estrategia que usan muchos desarrolladores de web es unas imágenes que entran en las resoluciones más pequeñas de pantalla y centrarlas en la pantalla. Añadiendo lo tags `<CENTER>` y `</CENTER>` rodeando el tag que muestra la imagen, el gráfico se verá correctamente en todas las resoluciones, incluyendo las más grandes, ya que se centrará y no parecerá que hay un espacio en blanco.

Ahora hay que ver qué ancho y alto vamos a utilizar para nuestros gráficos de web. Diferentes tipos de gráficos requieren diferentes alturas y anchuras.

Tipo de imagen	Coordenadas de altura y anchura (pixels)
Icono pequeño	25 x 25
Icono medio	40 x 40
Icono grande	60 x 60
Barra horizontal	10 x 500
Gráfico de cabecera	150 x 600
Anuncio en web	300 x 72
Logotipo o fotografía	300 x 400

Es importante destacar que estas medidas son sólo una orientación y que dependerán de las necesidades que tengamos en nuestro sitio web.

El tamaño de la imagen es proporcional al tamaño del fichero. De todas maneras, el tamaño del fichero depende también y en gran medida del número de colores usados en la imagen, el tipo de fichero que usemos (GIF o JPEG), y de la complejidad de la imagen y los diseños que tenga.

5. Elegir un número correcto de colores

El número de colores disponibles tiene un impacto directo en cómo veremos la imagen y en el tamaño del fichero. Debemos elegir un tipo de imagen con muchos colores sólo si realmente los necesitamos porque el tamaño del fichero crece a la vez que el número de colores.

Tipo de imagen	Cuándo utilizar esta opción
2 colores (1-bit)	Permite sólo dos colores (blanco y negro). No se permiten ni sombras ni grises, pero las imágenes en este formato son extremadamente pequeñas y eficientes.
16 colores (4-bits)	Windows originalmente soportaba sólo 16 colores. Estos 16 colores cubrían la mayoría del arco iris y se convirtieron en los colores por defecto en la mayoría de las aplicaciones y gráficos. Algunas imágenes impresionantes pueden llegar a crearse con estos colores. Sólo el formato GIF soporta la paleta limitada a 16 colores. El formato de imágenes JPEG automáticamente permite 16.7 millones de colores sin atender al número que seleccionemos.
256 grises (8-bits)	El número máximo de sombras que los GIF pueden soportar, esta opción proporciona mayor flexibilidad que sólo el blanco y negro ofreciendo 256 tonos de grises. No hay diferencia entre usar 256 tonos de grises y 256 tonos de color.
256 colores (8-bits)	El estándar de 256 colores que la mayoría de los GIF usan, será el que generalmente escojamos para hacer gráficos GIF pequeños. 256 colores será el máximo de colores que utilicemos a no ser

Tipo de imagen	Cuándo utilizar esta opción que utilicemos fotografías escaneadas o utilicemos opciones avanzadas de los programas de gráficos. Los 256 colores usados son los mismos que encontraremos en la configuración por defecto en la mayoría de monitores VGA.
16.7 millones de colores (24-bits)	Con 16.7 millones de colores, nunca tendrás que usar el mismo color dos veces. Esta opción se utiliza cuando planeas grabar tu imagen en formato JPG. Algunas de las opciones avanzadas de algunos programas de retoque fotográfico, requieren que utilicemos millones de colores, dado que se mezclarán cientos de colores automáticamente.

En general, cuando creamos una imagen GIF elegimos 256 colores, y cuando creamos una imagen JPEG, elegimos 16.7 millones de colores. Las imágenes JPEG están optimizadas para 16.7 millones de colores y tienden a ofrecer mejores tamaños de fichero cuando usas muchos colores diferentes en una imagen. Después, siempre puedes decrementar o incrementar el número de colores disponibles en tu imagen. El número de colores que usas en un gráfico de web es sólo tan útil como la resolución de la gente que vaya a tu web se lo permita. Digamos que eliges una imagen que usa 16.7 millones de colores, pero un visitante que tiene una VGA a 256 colores entra en nuestra web. Verá nuestro gráfico a 16.7 millones de colores como un gráfico de 256 colores.

6. Formatos GIF

Para grabar nuestro gráfico en formato GIF, elegiremos GIF-CompuServe en el cuadro de diálogo. Aunque los cuatro subtipos son muy similares entre sí, cada subtipo afecta a la forma en que veremos el gráfico en una página web. Aquí tenemos una pequeña descripción de los cuatro subtipos:

Version 87a NonInterlaced² - El estándar original GIF desarrollado por CompuServe en 1987. Comúnmente conocido como el *CompuServe GIF Standard*, es ampliamente utilizado en CompuServe y en Internet y fue el primer formato de gráficos soportado por los browsers de WWW. Es la configuración por defecto en que los ficheros GIF se han de grabar; el fichero es decodificado y mostrado normalmente en el browser.

Version 87a Interlaced³. Una pequeña variación del formato original, entrelazando imágenes, permite a los browsers de WWW

² NonInterlaced = No Entrelazado.

³ Interlaced = Entrelazado.

mostrar la imagen en pasos, cada paso nos trae la imagen con más detalle. Usar imágenes entrelazadas incrementa un poco el tamaño del fichero, pero es una opción excelente para la mayoría de desarrolladores de web.

Version 89a NonInterlaced. En 1989, dos años después de que el estándar original GIF fuese presentado, el estándar fue mejorado para aumentar la flexibilidad, compresión y eficiencia. También nuevo en la versión 89a fue la posibilidad de crear animaciones GIF.

Version 89a Interlaced. Igual que la versión 89a, pero con posibilidades de entrelazado. Este estándar es la opción más popular entre los desarrolladores de web hoy, porque ofrece la mejor funcionalidad GIF y flexibilidad.

Para todos nuestros gráficos, elegiremos Version 89a Interlaced. Es la mejor opción para desarrolladores de web porque las imágenes entrelazadas ofrecen flexibilidad cuando se utilizan en las páginas web. Las imágenes entrelazadas se cargan en varios pasos, de manera que en cada paso están más claros y más detallados, de manera que permiten ver poco a poco un mejor entorno de la página web más que tener que esperar a que se descargue completamente.

6. Formatos JPEG


El otro formato de imágenes disponibles es el JPEG. Recuerda que el formato de imagen JPEG usa 16.7 millones de colores pero normalmente es más eficiente al comprimir grandes imágenes que usan muchos colores, como las fotografías.

Similar a los GIF, el tipo de ficheros JPEG tiene también dos subtipos disponibles - estándar y progressive. El JPEG Estándar usa algoritmos de compresión muy buenos y muestra imágenes en una página web como un GIF no entrelazado. Tal y como la imagen va llegando, tu browser la va mostrando de arriba abajo.

Los Progressive JPEGs, por otra parte, son similares al formato entrelazado. Los Progressive JPEGs se muestran en múltiples pasos. Desafortunadamente los Progressive JPEGs no son tan ampliamente soportados como los formatos múltiples GIF. Actualmente, sólo Netscape 2.0 e Internet Explorer 3.0 soporta los JPEG progresivos; otros browsers más antiguos no pueden mostrar este tipo de imágenes. Cuando salvemos una imagen en formato JPEG utilizaremos sólo el formato Standard JPEG. Los Progressive JPEGs también ofrecen un 5% de mejora en el tamaño de los ficheros respecto a los JPEG Estándar.


8. Resumen de utilización de Paint Shop Pro





 Esta herramienta nos permite ampliar el tamaño de visualización de los gráficos al hacer clic sobre la zona que queramos ampliar con el botón izquierdo del ratón, y se reduce el tamaño al hacer clic con el botón derecho.


 Nos permite mover una selección que previamente hemos realizado.

 Seleccionaremos con esta herramienta áreas rectangulares con forma de elipse, cuadrado o círculo.

 Se trata de otra herramienta de selección que nos permite seleccionar un área poligonal o a "mano alzada".

 Permite realizar una selección basándonos en los colores, brillos o apariencias parecidas. Además, nos permite selección un nivel de tolerancia con lo que podemos hacer selecciones muy interesantes.

 Con esta herramienta podemos seleccionar un color de los existentes en el gráfico, sin que por ello tengamos que buscarlo en la tabla de colores, sino que lo cogemos directamente del gráfico.

 La brocha nos permite lo típico que podemos hacer, es decir, pintar al igual que lo podríamos hacer en el PaintBrush, pero con muchas más opciones como pueden ser la forma del pincel, el tipo de pincel (fluorescente, lápiz, brocha, etc), la textura que queremos simular que tiene el papel, etc. Si utilizamos el botón izquierdo pintamos con el color

primario y si pintamos con el botón secundario, utilizamos el color secundario.



Esta herramienta no he tenido oportunidad de verla en ninguna aplicación, y me parece interesante aunque dudo un poco de su utilidad. Se trata de una herramienta de clonación (si traducimos literalmente los manuales). Para utilizarla tenemos que coger la herramienta, hacemos clic con el botón secundario sobre el área que queramos clonar y posteriormente nos vamos donde la queremos copiar y la vamos pintando con el botón primario... se ve mucho mejor en un ejemplo práctico.



Simplemente nos permite reemplazar un color con otro, incluyendo opciones de texturas y demás. Siempre reemplaza el color de segundo plano con el color de primer plano.



Disponemos de muchos filtros que podemos aplicar en las zonas que deseemos mediante una brocha, permitiendo también incluir efectos de textura del papel.



El efecto que produce es un deshacer.



Simplemente un spray que también tiene las típicas opciones de texturas de papel, tamaño etc.



Rellenamos con efectos de degradados de color y podemos elegir si queremos que el relleno se realice por similitud de colores, de brillo, etc.



Herramienta para escribir textos, que podemos utilizar con todos los tipos de letra de Windows.



Para dibujar líneas a las que podemos colocar el grosor que deseemos.



Permite dibujar figuras de varios tipos con opciones de relleno.

Filtros, deformaciones y efectos especiales

Ya estés haciendo tus gráficos "a mano", modificando trabajos existentes, o escaneando tus fotografías favoritas, el software de gráficos como Paint Shop Pro puede hacer el trabajo más duro por ti.

9. Filtros para imágenes

Un buen fotógrafo o profesional de la publicidad puede hacer correcciones de color muy impresionantes en una habitación oscura tradicional. Otras formas de mejorar las imágenes, son imposibles de conseguir sin un ordenador. Los filtros de imágenes basados en una

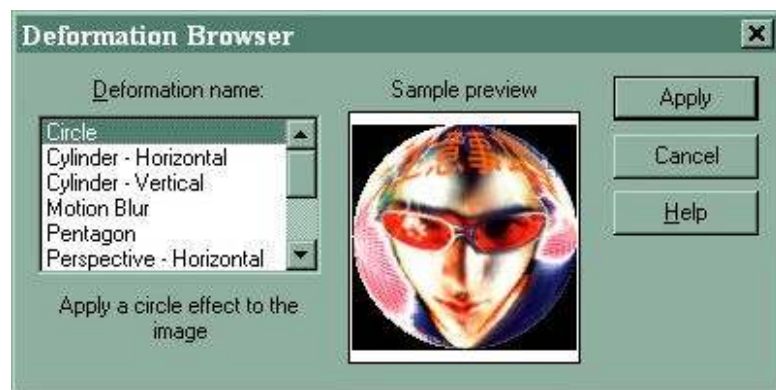
técnica matemática denominada "convolución" pueden parecer casi mágicos por la habilidad que nos dan de conseguir detalle, suavizar o quitar los bordes., y producir automáticamente efectos impresionantes como el emboss (repujado). De hecho, el control mediante cuadros de diálogo de los filtros para imágenes de Paint Shop Pro hace la elección y aplicación de filtros una tarea rutinaria.

El filtro emboss es especialmente popular en los desarrolladores de web para crear fondos y otros gráficos que parecen destacar de la página. Haciendo el color principal de una imagen repujada transparente, podemos combinar el repujado con un fondo para conseguir efectos sofisticados de manera que ninguno se superponga al otro.

Es notable que los filtros edge y emboss dan mejores resultados en imágenes que no han sido escaneadas con un escáner de mano.

10. Deformaciones para imágenes

Retocar imágenes requiere mucho trabajo. A pesar de eso, puede ser muy divertido deformar imágenes. Así como con los filtros, Paint Shop Pro te da un asistente para deformaciones.



Cuando eliges una deformación en el browser y pinchas en "Apply", tienes una caja de diálogo que te permite ajustar la configuración del efecto para una deformación particular y previsualizar los resultados en la imagen completa antes de realizar la deformación.

11. Combinar y solapar las imágenes

Algunos de los efectos gráficos más útiles puede conseguirse al combinar dos imágenes con "image arithmetic". Para combinar imágenes de muchas maneras diferentes, utiliza la opción de menú Image | Image Arithmetic. Por ejemplo, puedes sumar dos imágenes en una, haciendo que una superponga a la otra con una transparencia parcial.

Aquí tenemos un recorrido rápido por el cuadro de diálogo que aparece en Image Arithmetic:

- **Elige las imágenes que quieras combinar desde los desplegables image #1 y image #2. Para obtener mejores resultados asegúrate de que las dos imágenes tienen el mismo tamaño.**
- **Elige una opción aritmética como Sumar o Multiplicar de las selecciones de función.**
- **Para las imágenes en color, puedes elegir entre trabajar con los canales rojo, verde o azul sólo. Normalmente, de todas maneras, debes elegir Todos los canales para trabajar con todos los colores de la imagen a la vez.**
- **En los Modificadores, introduce un Divisor y Sesgo (Bias) para la operación que tengas en mente.**
- **Casi siempre querrás elegir opciones Clip de color. Si no seleccionamos esta opción, el blanco super-brillo se convierte en negro o gris, y los negros oscuros se convierten en grises o blancos. Esto puede crear efectos interesantes, pero es difícil de predecir y controlar.**

Cuando pinchas en OK, Paint Shop Pro va píxel por píxel en la primera imagen y los suma (o multiplica, o resta, etc) con el valor correspondiente del píxel de color de la segunda imagen. El resultado se divide entre el número que hemos puesto en la caja Divisor, y se añade el sesgo a ese número.

La suma de imágenes es relativamente fácil de describir, pero las otras funciones de imagen aritméticas ofrecen posibilidades creativas que no podríamos explicar con palabras.

12. Efectos de color

Paint Shop Pro, como la mayoría del resto de programas de gráficos serios, ofrece un número de comandos de manipulación de color. Estos son fáciles de usar una vez que se ha entendido lo que hacen. Pero, si no eres un artista gráfico con experiencia, la simple lectura de los nombres que veas en el comando de menú Color, puede que no te de mucha idea de para qué se usan estos comandos.

Esta referencia rápida debe darte información suficiente para comenzar a trabajar con estos comandos de color. También te daremos algunos trucos para crear páginas web.



12.1. Grey Scale y Colorize

Colors | Grey Scale simplemente convierte una imagen en tonos de grises, lo que permite dar un look de TV antigua o retro años 50 a

nuestra web. **Colors | Colorize** tinta una imagen en grises o en color, manteniendo el brillo relativo en la imagen. Especificamos el Hue (tinte) como un número entre 0 (rojo) y 255 (violeta), que van en el orden del arco iris como se muestran en la tableta de elección de color en la derecha del Paint Shop Pro. También podemos elegir la saturación entre 0 (grises) y 255 (colores brillantes).

La función **Colors | Colorize** es muy útil para pintar fotos, fondos y gráficos existentes con el esquema de colores que hayamos elegido para nuestra web. También puedes crear una fotografía a la antigua eligiendo **Colors | Gray Scale** y utilizando **Colorize** para pintar la imagen con marrones o azules.

12.2. Negative y Solarize

Colors | Negative Image reemplaza cada color de la imagen con su opuesto, tal y como haría el negativo en una fotografía.

Colors | Solarize también es un efecto de la fotografía química (tradicional). En la fotografía, este proceso nos lleva a tomar una fotografía y la exponemos intencionalmente a la luz solar por un período controlado de tiempo. Algunos fotógrafos de la mitad del siglo veinte se hicieron muy populares por "solarizar" sus imágenes para el resultado que podíamos apreciar en obras como las de Andy Warhol. Ahora puedes experimentar este efecto mucho más fácilmente que con la fotografía tradicional, y controlar el umbral de la imagen.

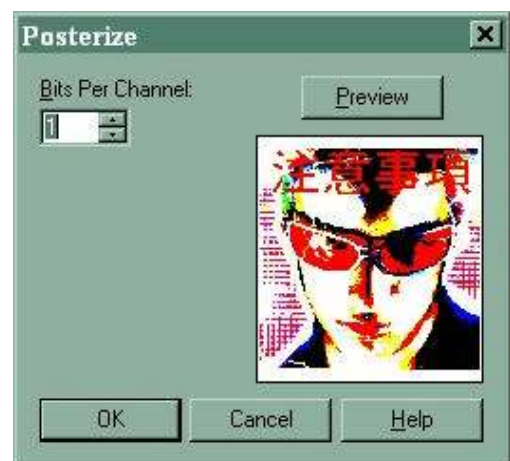


En las páginas web, **Negative Image** y **Solarize** deben ser utilizados con mucha prudencia a no ser que queramos crear una página web con un look diferente e incluso puede que en ocasiones nos

queden unos colores muy oscuros o dramáticos.

12.3. Posterize y Decrease Color Depth

El comando **Colors | Posterize** debe su nombre de una antigua técnica usada para reducir el número de colores



en una imagen de manera que pudiera ser imprimida a bajo precio en un póster, en los días antes de que el proceso de enlazado de cuatro colores fuese común. Esencialmente, indicamos el número de colores que queremos utilizar en nuestra imagen y Posterize elige los mejores colores que coincidan con los colores dominantes en nuestra imagen.

De todas maneras, entender y utilizar Posterize es un poco más difícil que esta explicación simplificada porque hay que especificar el número de bits utilizados para describir cada canal de color en la imagen, además de especificar el número actual de colores.

Especificar el número de bits por canal de color puede ser confuso si no estás al tanto de la teoría de colores y matemática de computadoras, por tanto ponemos esta tabla para que se entienda mucho más fácilmente:

Bits por canal	Número de colores por canal	Número total de colores distintos
1	2	8
2	4	64
3	8	512
4	16	4096
5	32	32768
6	64	262144
7	128	2097152
8 (color pleno)	256	16777216

Para las escalas de grises hay sólo un canal, por lo que el número de posibles escalas de grises corresponderá con la columna del centro de la tabla.

También puedes decrementar el número de colores en tu imagen seleccionando Colors | Decrease Color Depth. Esta opción usa un mapeado de color y técnicas que Posterize para conseguir una aproximación mejor a la apariencia de la imagen original. También puede ser utilizado para efectos únicos.

13. Efectos especiales

El menú de Paint Shop Pro Image | Effects es un tesoro a la hora de enriquecer nuestras páginas web.

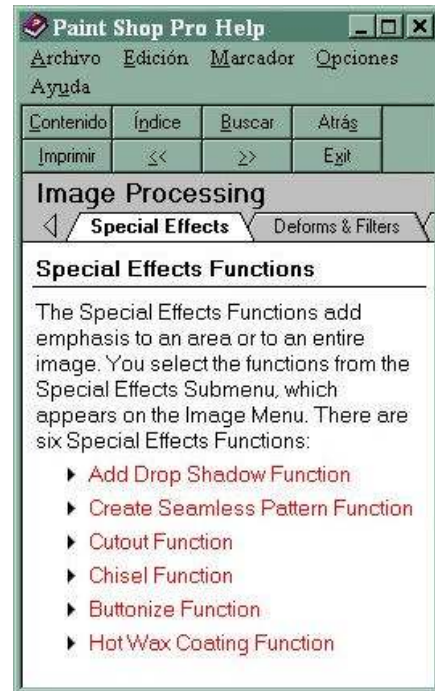
13.1. Drop Shadows y Highlights

Efecto
Drop Shadows

Uno de los efectos más populares y utilizados es el "drop shadow" - un oscurecimiento del área que está directamente debajo de un gráfico o texto

para hacer que parezca que está flotando sobre la página. Y al contrario que el resto de efectos de gráficos que hemos explicado, este efecto podemos aplicarlo tanto como queramos, porque no se va a hacer pesado como otros efectos que hemos visto.

Afortunadamente, Paint Shop Pro hace la realización de sombras increíblemente fácil. Puedes añadir una sombra debajo de cualquier texto o región seleccionada con el comando **Image | Special Effects | Add Drop Shadow**. La caja de diálogo de Drop Shadow nos permite elegir el color de la sombra y nos da mucho control también sobre otras partes del gráfico. Puedes seleccionar la Opacidad desde 1 (casi transparente) hasta 255 (sólida). La opción **Blur** nos permite decir cuantos pixels difuminarán la sombra que llevará desde 1 (no difuminado) hasta el 36 que puede dar una sensación de fluorescencia si usas un color brillante.



13.2. Crear Botones 3D

Una vez que tenemos un título impresionante con un a sombra debajo, naturalmente querremos crear algunos botones 3D para establecer enlaces entre nuestras páginas web. Otra vez más, Paint Shop Pro lo hace en un momento. Seleccionamos un área rectangular con la herramienta de selección (o la imagen completa) y elegimos **Image | Special Effects | Buttonize** para convertir la selección en un botón 3D en un instante.

La altura del botón se elige con **Edge Size** como un porcentaje de la altura total del botón. También puedes elegir entre un botón con ejes transparentes o ejes sólidos. En cualquier caso, el botón se sombreadá con una combinación del color de primer plano, la imagen existente y sombras de grises para crear el efecto 3D.

Chisel digamos que hace algo parecido a un cincelado, y **Hot Wax Coating** tinta y hace **Chisel** a la misma vez. La opción final, **Cutout**, crea un efecto de biselado en los ejes de la selección de manera que parece que hay un socavón en la imagen.

13.3. Hot Wax y Tinting

El último elemento de la lista de Efectos Especiales de Paint Shop Pro es un poco diferente del resto, "Hot Wax Coating" (cubrir con cera caliente), no suena a algo que podríamos hacer con nuestro ordenador, a no ser que utilicemos un programa de gráficos. Aunque no lo parezca es una herramienta muy útil para las páginas web, porque combina un montón de tareas simples en una sola.

La matemática que hay bajo Hot Wax es interesante y el efecto también. Básicamente, tinta la selección actual con el color de primer plano, aunque también mejora los ejes, aumenta el contraste, y oscurece los brillos de una manera que sería difícil de conseguir utilizando herramientas simples.

Aunque aquí hayamos dado unas pinceladas a la forma de crear efectos en los gráficos con Paint Shop Pro, está claro que Paint Shop Pro también soporta los filtros añadidos estándar que son creados por muchas de las compañías de gráficos de todo el mundo. Estos filtros añadidos, de genios de los gráficos como los creadores de las Power Tools de Kai, Alien Skin Black Box y Adobe Photoshop, pueden dar un nuevo aire a nuestras páginas web y hacer que queden muy bien.

14. Hacer gráficos eficientes

En las anteriores secciones hemos aprendido a hacer y manipular todo tipo de imágenes ya sea desde digitalizaciones directas de escáner o bien desde un gráfico simple que posteriormente vamos complicando con ayuda de las opciones avanzadas de Paint Shop Pro.

Esta parte del curso toma un punto de partida diferente a la hora de trabajar con gráficos. Aprenderemos muchas técnicas para hacer que los gráficos además de ser estéticamente correctos (al menos), sean también pequeños y eficientes. A partir de que tomamos que todo fichero que queramos ver con nuestra web necesitará cargarse, tenemos que tener en cuenta que influirá directamente en la comodidad de visionado de nuestra web. Nadie quiere esperar demasiado tiempo a la hora de ver una página web.

En esta sección presentaremos algunas estrategias efectivas que pueden ayudarte a la hora de realizar gráficos pequeños manteniendo una calidad de detalles aceptable. Desafortunadamente la mayoría de estos beneficios vienen de parte de los colores de la imagen y la resolución. Mostraremos cómo regular estos sacrificios para conseguir la mejor calidad de imagen y las imágenes más eficientes (orientadas a web).

14.1. ¿Porqué utilizar gráficos eficientes?

Uno de los principales obstáculos que tenemos los usuarios de web hoy día es la cantidad de tiempo que tarda en llegar una página web. Cuando visitas una página web, hay que esperar hasta que el texto y las imágenes sean transferidas o "descargadas", desde internet hasta nuestro ordenador personal. El tiempo que tarda en descargarse depende del tipo y velocidad de nuestra conexión. La mayoría de gente que se conecta desde casa, utiliza un módem para el WWW, aunque algunas empresas tienen conexiones directas a Internet.

La velocidad del módem indica la razón a la que los gráficos pueden descargarse. Las velocidades populares de los módems van en rangos desde 14.4 a 28.8 miles de bits por segundo, pero aún queda gente que utiliza módems incluso de velocidades menores. De hecho, las nuevas técnicas de compresión pueden incrementar la velocidad de nuestro módem. Cuanto más rápido es el módem, más grandes pueden ser los gráficos.

Aunque se trate del mayor rango de baudios tarda algunos segundos en cargar y visualizar las imágenes en las web muy grandes. De todas maneras, lo más interesante en el diseño de una página web es conseguir que se cargue en el menor tiempo posible. Reduciendo el tiempo que tarde en cargarse nuestra página, conseguiremos mejor respuesta por parte de nuestros usuarios y probablemente volverán a una posterior visita.

Todos esperan tardar un poco en cargar los datos al visitar una página web, pero nadie quiere esperar 30 segundos para cargar cada página, ver el contenido y pinchar para ir a otra página. Es posible que si el tiempo de espera es demasiado largo, la gente termine por desesperarse, pulsar el botón de stop y continuar viendo otras páginas, por supuesto, diferentes a la nuestra.

Conseguir unos gráficos pequeños y eficientes es necesario, pero a la vez todo un reto.

14.2. Recorte, redimensionado y reducción de imágenes

Una de las formas más populares para reducir el tiempo total de descarga de una página web es reducir el área actual en pixels de la imagen que estamos descargando. Puedes ahorrarte mucho tiempo redimensionando o recortando las imágenes de manera que se envíe una imagen mucho más pequeña en lugar de la original. Los visitantes pueden pinchar en un enlace de hipertexto si quieren ver una versión a tamaño completo de la imagen.

Esta sección explica dos de las mejores maneras para reducir el tamaño de las imágenes en tu página web. Usaremos estos métodos principalmente cuando utilicemos fotografías e ilustraciones en nuestro sitio web. No sólo aumentaremos el rendimiento a la hora de cargar nuestras páginas web, sino que el diseño de páginas con gráficos pequeños es mucho más fácil.

14.2.1. Redimensionar una imagen

El error más grande que puede hacer un desarrollador de web es usar una imagen que sea muy grande innecesariamente.

El primer paso a la hora de redimensionar una imagen es cargar la imagen original con Paint Shop Pro. Elegimos File | Open de la tabla de menú y seleccionamos la imagen con la que vamos a trabajar. Para redimensionar imágenes con Paint Shop Pro, elegimos Image | Resize de los menús para obtener el cuadro de diálogo de Paint Shop Pro.

Ahora podemos elegir entre algunos tamaños estándar o introducir un tamaño que nosotros elijamos.

¡¡Atención!! Tenemos que asegurarnos de que no sobrescribimos la imagen original por error. En su lugar, la salvaremos a un nuevo fichero con el comando File | Save As. Renombraremos la imagen con un nombre similar pero descriptivo de manera que seamos capaces de diferenciar cada uno de ellos claramente.

Podemos redimensionar nuestros gráficos a casi cualquier tamaño imaginable. No importa si hacemos varias pruebas con diferentes tamaños hasta encontrar el que nos interesa.

Como puedes imaginar, el tamaño en pixels es directamente proporcional al tamaño del fichero.

También podemos redimensionar las imágenes JPEG tan fácilmente como las GIF. Tan sólo debemos recordar que el formato JPEG no reconoce algunas opciones especiales de los GIF como la transparencia y el entrelazado. De todas maneras, podemos de esta manera (pasar de GIF a JPEG) perder tamaño de nuestro fichero, lo que siempre está bien.

El uso de imágenes más pequeñas en nuestra página web requiere un poco de más creatividad. Es una buena idea aprender formateado de texto en HTML para cambiar la apariencia de nuestra página web sin tener que desarrollar imágenes enormes.

14.2.2. Haciendo thumbnails (reducciones)

Cuando reducimos el tamaño en pixels de los gráficos de nuestra web, disminuimos significativamente la porción de tiempo que los visitantes tardan en cargar nuestros gráficos. Desafortunadamente, redimensionar a una imagen más pequeña a veces hace que nuestros gráficos sean más difíciles de ver y menos vistosos para la gente que en realidad lo que quiere ver es una imagen a tamaño completo. A partir de que la imagen es físicamente más pequeña, hay que observar que los detalles seguramente se perderán.

Para compensar este problema potencial, la mayoría de los sitios web utilizan un proceso denominado **thumbnailing**, que da a los visitantes la oportunidad de ver tanto las versiones grande como pequeña de una fotografía. Thumbnail es un proceso en el que mostramos la imagen pequeña en nuestra página web, pero añadimos un link de hipertexto a la imagen grande. Esto permite a los visitantes ver la fotografía en su tamaño original y más grande pero sólo si ellos quieren verla.

Hacer thumbnails es fácil. Primero creamos la imagen grande. Entonces, de acuerdo con los pasos que hemos visto en las secciones anteriores, hacemos y salvamos una versión menor. Nombramos los ficheros de una forma lógica. No tiene que haber duda de qué fichero representa la imagen completa y cual es la versión thumbnail.

Normalmente, cuando añadimos una imagen a nuestra página web, usamos la siguiente línea de HTML:

```
<IMG SRC="perropequeno.gif">
```

De esta manera, y al usar thumbnails, queremos enlazar nuestra imagen menor con a grande. Para llevar a cabo esto, añadimos los tag **<A HREF>** y **** "rodeando" el tag original de la imagen:

```
<A HREF="perrogrande.gif"><IMG SRC="perropequeno.gif"></A>
```

Esta línea de HTML no sólo le dice a nuestro browser que muestre **perropequeno.gif** como parte de la página web, sino que también les dice a los visitantes que pueden pinchar en esta imagen. Cuando el visitante pincha en la imagen, ésta se descarga y es mostrada al usuario.

14.2.3. Recortar

Otra forma de reducir el tamaño de nuestro gráfico para web es recortarlo y mostrar una pequeña parte del original. El recorte de imágenes ha sido una herramienta en los escritorios de los editores,

diseñadores gráficos, etc. Usualmente, hay partes extra de la imagen que pueden cortarse. El resultado es una imagen más pequeña que sólo contiene el material interesante y útil.

Siguiendo con el mismo ejemplo canino, no necesitamos mostrar la imagen del perro completa sino que podemos darle una idea de qué aspecto tiene Bobby. Recordando la foto y utilizando sólo su cabeza, conseguimos un enorme ahorro de tamaño.

Paint Shop Pro tiene capacidad de hacer recortes. Usando el ratón, simplemente indicamos qué parte de la imagen debe ser salvada aparte. Para recortar una imagen seguiremos los siguientes pasos:

- 1. Cargar la imagen original en Paint Shop Pro usando el comando File | Open.**
- 2. Pinchamos el icono de selección de la tabla de herramientas de PSP. Esto nos permite seleccionar un área de nuestra imagen que recortaremos y salvaremos. Podemos seleccionar un área cuadrada, rectangular, circular o elíptica definiendo la forma en la paleta de estilos de PSP.**
- 3. Utilizando el ratón, elegimos la parte de la imagen que queremos recortar.**
- 4. Elegimos Image | Crop de la barra de menús. Paint Shop Pro se quedará con la imagen seleccionada y desechará el resto de la imagen original.**
- 5. Salvaremos nuestra imagen recortada con el comando File | Save As de manera que no sobrescribamos el gráfico original.**

Otra ventaja adicional de este método es que al no redimensionar el gráfico, el usuario ve todos los detalles del gráfico a la primera.

Tal y como veíamos en las imágenes redimensionadas, algunos desarrolladores de web también enlazan la imagen recortada con la imagen grande. Esto te permite que un recorte haga las funciones de un thumbnail de forma que los visitantes puedan ver la imagen completa.

15. ¿Cuántos colores están bien para una imagen GIF?

El número uno a la hora de reducir el tamaño de tus fichero GIF es reducir el número de colores que usamos en la imagen. En los ficheros GIF, el número de colores diferentes usados tiene una relación directa con el tamaño del fichero. Cuantos menos colores utilicemos, menor es el tamaño del fichero. Esto es diferente en las estructuras de ficheros JPEG, donde el fichero depende menos del número de colores utilizados, sino del nivel de compresión utilizado.

15.1. Cómo afectan los colores al tamaño de un fichero GIF

Actualmente, el tamaño del fichero no es completamente dependiente del número de colores utilizados como puedes imaginar. La situación de los colores en una imagen también afecta al tamaño de un fichero GIF, como pasamos a explicar.

De acuerdo con las especificaciones del formato GIF, una imagen se graba como una serie de líneas horizontales que van por la pantalla de izquierda a derecha. Comenzando con el primer pixel en la izquierda de la pantalla, la imagen graba las especificaciones de un determinado color, digamos que un azul. Siguiendo hacia la derecha, un pixel cada vez, el GIF asume que debe seguir manteniendo las especificaciones del color hasta que definamos un color diferente. De tal manera, si la línea completa es azul, sólo necesitaremos definir el color de la línea al principio. Normalmente, hay muchos colores diferentes en una sola línea de imagen. Cada vez que un color diferente necesita ser mostrado, esa información se salva en el fichero GIF. Una vez que la parte de la derecha de la imagen ha sido alcanzada el GIF empieza de nuevo, como una máquina de escribir, y comienza a definir la siguiente línea (pixel por pixel) de colores en la imagen.

Es fácil darse cuenta de que si sólo necesitamos un color para toda la imagen el fichero tenderá a ser menor porque no hay cambios de imagen. Es increíble que una imagen del mismo tamaño (en píxels) tenga un aumento tan enorme al aumentar el número de colores posibles en la imagen.

Desafortunadamente, esta estrategia sólo sirve en los ficheros GIF. Los ficheros JPEG siempre tienen 16,7 millones de colores disponibles y usan una estructura de fichero y esquema de compresión que es diferente que en los GIFs. Por otra parte un GIF grande que utilice muchos colores diferentes, mejorará significativamente cuando lo convirtamos en JPEG. Notaremos esto especialmente al escanear fotografías o ilustraciones para nuestra web, que siempre usa montones de colores y tendrán un cambio significativo de colores en ellos.

La compresión JPEG no es dependiente del número de colores. En su lugar, los ficheros JPEG utilizan un formato especial de compresión que puede perder detalle en ocasiones. En ocasiones, querrás grabar los gráficos en los dos formatos, tanto GIF como JPEG y ver cual representa mejor la imagen con una calidad de tamaño de fichero aceptable.

15.2. Reducir los colores

Ahora que hemos entendido la relación entre el número de colores utilizados y el tamaño resultante del fichero GIF, podemos pasar a un tema que resultará vital en algunas ocasiones. Reducir el número de colores y los cambios de color en una imagen puede hacer que tus ficheros ocupen un 75% menos.

La estrategia de reducción de color más popular es coger una imagen existente a 256 colores y transformarla en una de 16. PSP hace el mapeado de color y los cambios por ti. Esta estrategia debe ser utilizada con mucho cuidado. Algunas veces, reducir el número de colores degrada la apariencia de la imagen hasta el punto que no podemos utilizarla para nuestra página web. Tendrás que evaluar cada GIF por separado y comprobar los resultados por ti mismo/a.

Cuando reduces el número de colores, dices a PSP que transforme tu imagen de 256 colores en una que sólo utilice 16 separados y únicos. PSP trata de ajustar cada uno de los 256 colores originales con uno de los 16 que quedarán. Reduciendo el número de colores utilizados pierdes mucho detalle, pero el ahorro de espacio en cada imagen es tremendo.

El primer paso es darse una idea del número de colores en nuestro GIF. Cargamos nuestro GIF en PSP y elegimos Colors | Count Colors Used desde la tabla de menú. Una pequeña caja de diálogo aparecerá y mostrará el número de colores que se utilizan en nuestra imagen. Pincha en OK para salir de esa ventana.

Para reducir el número de colores en nuestro GIF, elegimos Colors | Decrease color Depth | 16 Colors (4 bits). No sólo tenemos menos colores de los que elegir, sino que se definen en 4 bits, la mitad de lo necesario para definir 256. La caja de diálogo de Decrease Color Depth aparece.

Pinchamos en OK para continuar. PSP automáticamente interpola nuestra imagen actual y muestra la nueva resultante del proceso.

Reducir el número de colores de 256 a 16 puede darnos una gran variedad de resultados según la imagen original.

Además, no es siempre una solución perfecta. En ocasiones perdemos detalles significativos y precisión que teníamos en la imagen original. Esto ocurre principalmente cuando un gran rango de colores diferentes se utilizan en una sola imagen. Ajustando 256 colores a 16 se convierte en una tarea difícil, PSP lo hace lo mejor que puede, pero en ocasiones, hay deformaciones.

Algunas veces, por supuesto, reducir el número de colores deforma nuestro original tanto, que no nos compensa la reducción en el tamaño del fichero.

16. Compresión JPEG

Similarmente, el formato JPEG nos permite un gran rendimiento consiguiendo también una reducción de tamaño. La mayor compresión, el menor tamaño. Los JPEG comprimidos tardan más en mostrarse en los browsers porque las imágenes deben descomprimirse antes de mostrarse. Además, comprimir las imágenes JPEG puede hacer que perdamos algo de calidad. Estas pérdidas normalmente no se notarán a no ser que tengamos una resolución enorme.

En PSP, la compresión de imágenes JPEG se selecciona cuando vamos a grabar nuestros ficheros. Abrimos cualquier imagen JPEG y elegimos File | Save As para hacer salir la caja de diálogo de Save As. Entonces, pinchamos el botón Options para sacar la caja de Preferencias de Fichero.

En la etiqueta marcada como Compression Level, elegimos el rango entre 1 y 99. El mayor número, la mejor compresión para nuestro JPEG y en consecuencia, el menor tamaño de fichero.

Cuando la imagen se salva, el nivel de compresión también se graba con la imagen. Pincha OK para volver a la caja de diálogo Save As. Desde aquí puedes especificar un nombre de fichero para tu imagen JPEG.

¡¡Atención!! En la caja de diálogo Save As, asegúrate de que el tipo de fichero que especificas es JPEG.

En general, los niveles de compresión por encima de 90 nos dan tantas interferencias que las imágenes se convierten en inútiles.

17. Usar Interlaced y Progressive

Hemos visto que los gráficos GIFs entrelazados se muestran en varios pasos, de manera que en cada paso son más detallados y claros.

Los GIFs entrelazados son útiles, aunque sea cuando descargamos un GIF enorme, porque puedes hacerte una idea general de cómo es la imagen a la vez que se va descargando y esto es muy útil para la gente con un módem lento.

El formato de fichero JPEG nos permite funcionalidades similares cuando grabamos un fichero en formato JPEG Progressive, aunque como no está muy extendido, utilizaremos el formato típico.

Salvar una imagen en un formato entrelazado o progressive es simple. Después de crear nuestra imagen, elegimos File | Save As para conseguir la caja de diálogo de Save As. Después de decidir entre JPEG y GIF, miramos las opciones en el Sub tipo.

Cuando grabamos un GIF, podemos elegir entre Entrelazado o No entrelazado. Con los JPEG podemos utilizar codificado Estándar o Progresivo. Como normal general, grabaremos las imágenes en Entrelazado o Progresivo cuando la imagen sea mayor de 15 K. Para los iconos pequeños, botones y barras, no tenemos que preocuparnos por grabar en formatos Entrelazados o Progresivos porque son tan pequeños que se cargan casi instantáneamente.

Salvar en formato Entrelazado o Progresivo hace la imagen aproximadamente un 10% mayores que los originales, pero el beneficio es mucho mayor cuando grabamos imágenes mayores. Permitiendo a los visitantes ver unas líneas generales de una imagen conforme se va descargando aumenta la posibilidad de uso para una página web porque los visitantes pueden comenzar a leer la información en la página antes de que la imagen se descargue completamente.

18. Gif animados

Los gif animados son simples sucesiones de gráficos gif ya sean o no entrelazados. Nos centraremos en el uso de la herramienta de creación de gráficos animados Microsoft Gif Animator, por ser la más compacta y fácil de usar, aunque también nos serviremos de otras herramientas para optimizarlos y para hacerlos más fácilmente.

Podemos crear o modificar una animación y grabarla para usarla en nuestro proyecto. Podemos añadir tantas imágenes como nos permite la memoria del ordenador.

GIF Animator contiene una barra de herramientas, una columna de visualización de las imágenes de la animación, una barra de scroll y tres pestañas: la de opciones, la de animación y la de imagen.

- **La pestaña de Opciones controla la forma en la que GIF Animator manejará nuestros ficheros.**
- **La pestaña de Animación controla las características de nuestra animación.**
- **La de Imagen controla las características de cada imagen individual que utilizamos en la animación.**

Podemos añadir imágenes a la columna de la animación utilizando arrastrar y soltar, pegando una imagen desde el portapapeles o abriendo un GIF existente desde Gif Animator. Utilizamos Import Color

Palette en las Opciones antes de añadir imágenes a nuestra animación. Usamos la barra de desplazamiento para ver todas las imágenes en la animación que estamos utilizando. Las imágenes se insertan antes de la que tenemos en selección.

18.1. Barra de herramientas

Aquí mostramos las opciones de que disponemos en la barra de tareas, de izquierda a derecha:

- **Nuevo: Crea un nuevo fichero.**
- **Abrir: Abre un fichero existente. Si usas esta opción cuando ya tienes un fichero abierto, GA avisa de que los cambios se perderán si no se graban y te permite grabarlos.**
- **Salvar: Salva los cambios al fichero activo.**
- **Insertar: Inserta un GIF adicional en la animación actual. El fichero se inserta antes de la imagen seleccionada.**
- **Salvar como: Salva los cambios a un fichero diferente del por defecto.**
- **Cortar: Quita la imagen seleccionada y la pasa al portapapeles.**
- **Copiar: Duplica la imagen seleccionada al portapapeles.**
- **Pegar: Coloca una imagen copiada o pegada desde el portapapeles al punto de inserción**
- **Borrar: Borra la imagen seleccionada sin copiarla al portapapeles.**
- **Seleccionar todo: Selecciona todas las imágenes en la animación.**
- **Mover arriba y mover abajo: Posiciona la imagen seleccionada un punto más arriba o un punto más abajo.**
- **Previsualizar: Muestra los resultados de nuestra animación, sin salvarla.**
- **Ayuda: Muestra un pobre sistema de ayuda.**

18.2. Pestaña Opciones

La pestaña Opciones de GA permite establecer que paleta del Animator se utiliza para representar las imágenes en la animación y cómo se representan los colores en la imagen salvada.

Thumbnails reflect Image Position: Para ver cada imagen en el espacio de animación que elegimos en Animation en lugar de cómo una imagen completa.

Main Dialog Window Always on Top: Para que GIF animator se mantenga siempre encima de todas las ventanas. Esto deshabilita el arrastrar y soltar.

Import Color Palette: Permite elegir entre la paleta del browser, que proporciona un ajuste directo a los browsers más comunes, o una

paleta óptima que podemos especificar pinchando en la caja de diálogo para localizar el fichero .PAL de Windows que quieres utilizar.

Import Dither Method: Permite elegir el método de escritura: solid, pattern, random y error diffusion. Utilizaremos siempre Solid.

18.3. Pestaña Animación

La pestaña animación de GA proporciona control sobre el tamaño, duración y transparencia de una animación.

Animation Width y Height: Nos permite decir el espacio horizontal y vertical, respectivamente, en que la animación se ejecuta.

Image Count: Muestra el número de imágenes en la animación actual. Más imágenes con pequeños movimientos proporcionan visionados más suaves, pero crean ficheros grandes y que tardan en descargarse.

Looping: Si queremos que se repita la animación.

Repeat Count: Cuántas veces queremos que se repita.

Trailing Comment: Comentarios que podemos insertar sobre el copyright, autor, etc.

18.4. Pestaña Imagen

Controla las características individuales de cada una de las imágenes en la animación.

Image width e Image height: Muestra la anchura y altura de la imagen seleccionada.

Left y Top: Nos permite especificar la posición desde los ejes izquierdo y superior hasta la situación de la imagen.

Duración (1/100s): Periodo de tiempo en centésimas de segundo que vamos a estar viendo la imagen. Podemos hacer efectos de comienzo y pausa.

Método de desdibujado: Undefined (no hace nada con el fondo antes de mostrar la imagen siguiente), Leave (Le dice al browser que deje el gráfico anterior y luego dibuja el nuevo), Restore Background (Redibuja el fondo original), Restore Previous (Redibuja la imagen previa),

Transparencia: Si queremos especificar un color que no se mostrará en la animación.

Transparent Color: Aparece una paleta en la que podemos elegir un color que GIF Animator utilizará como la porción transparente de la imagen. Puedes elegir sólo un color de transparencia.

Comments: Permite insertar comentarios a esta imagen en concreto.

19. Recursos sobre gráficos en la web

<http://www.jasc.com>
<http://galadriel.ecaetc.ohio-state.edu/tc/mt>
<http://web3d.asymetrix.com>
<http://www.asymetrix.com>
<http://www.ct.ebt.com/figleaf.html>
<http://www.mindworkshop.com/alchemy/alchemy.html>
<http://www.cis.ohio-state.edu/hypertext/faq/usenet/jpeg-faq/part2/faq.html>
<http://home.netscape.com>
<http://www.eu.microsoft.com/ie>
http://www.cc.ukans.edu/anout_lynx/about_lynx.html
<http://www.politicsnow.com>
<http://www.airplane.com>
<http://www.compuserve.com>
<http://www.cs.brown.edu/people/oa/Bin/skeleton.html>
<http://watt.emf.net/wm/paint/auth/michelangelo>
http://www.yahoo.com/Computers_and_Internet/Multimedia/Pictures
<http://www.kodak.com>
<http://www2.cybernex.net/~jen/webpages/bullets/bullets.html>
<http://www.inin.co.uk/images/gl/index.html>
<http://www.scs.unr.edu/veronica.html>

D. Javascript

1. Introducción
2. Primer programa
3. Sintaxis
4. Estructuras de control
5. Objetos
 - 5.1. Objeto window.
 - 5.2. Objeto document.
 - 5.3. Objeto history
 - 5.4. Objeto form
 - 5.5. Objeto Date
 - 5.6. Objeto Math
 - 5.7. Objeto Navigator
 - 5.8. Objeto Screen
6. Eventos
 - 6.1. Lista de eventos
7. Trucos
 - 7.1. Crear nuevas ventanas
 - 7.2. La barra de estado
 - 7.3. Volver a la página anterior
 - 7.4. Tratamiento de imágenes
8. Cookies
 - 8.1. Funciones básicas
 - 8.2. Ejemplo
9. Futuro
 - 9.1. Ficheros .js
 - 9.2. HTML Dinámico

D. JavaScript

1. Introducción

JavaScript, al igual que Java o VRML, es una de las múltiples maneras que han surgido para extender las capacidades del lenguaje HTML. Al ser la más sencilla, es por el momento la más extendida.

Antes de nada conviene aclarar un par de cosas:

- 1. JavaScript no es un lenguaje de programación propiamente dicho. Es un lenguaje *script* u orientado a documento, como pueden ser los lenguajes de macros que tienen muchos procesadores de texto. Nunca podrás hacer un programa con JavaScript, tan sólo podrás mejorar tu página Web con algunas cosas sencillas (revisión de formularios, efectos en la barra de estado, etc.) y, ahora, no tan sencillas (animaciones usando HTML dinámico, por ejemplo).**
- 2. JavaScript y Java son dos cosas distintas. Principalmente porque Java sí que es un lenguaje de programación completo. Lo único que comparten es la misma sintaxis.**

Este texto y en este cursillo se mostrarán una parte de las potencialidades del JavaScript. No pretendo hacer aquí una guía completa, sino mostrar una pequeña introducción. Para tener una guía de referencia es mejor acudir a la que ofrece Netscape⁴ que, al fin y al cabo son los creadores del invento.

Existen tres versiones de JavaScript. Casi todo lo que hay en este cursillo funciona con la versión 1.0 que nació con Netscape Navigator 2.0. Todo aquello que requiera versiones 1.1 o 1.2 se indicará.

El Microsoft Internet Explorer⁵ soporta el Javascript. No muy bien, pero lo soporta. La versión 3.0 del browser interpreta Jscript, que es similar al JavaScript 1.0 pero con algunas diferencias para provocar ciertas incompatibilidades. El Jscript 3.0 del Explorer 4 se comporta de modo parecido, admitiendo JavaScript 1.1 con cierta fiabilidad, aunque no plena, y algunas características del 1.2.

A partir de ahora suponemos que si has llegado hasta aquí, conoces los contenidos de los temas anteriores, y especialmente los del bloque B.

⁴ El manual aparece como "Javascript Handbook" y puede encontrarse desde la web principal de Netscape <http://www.netscape.com>

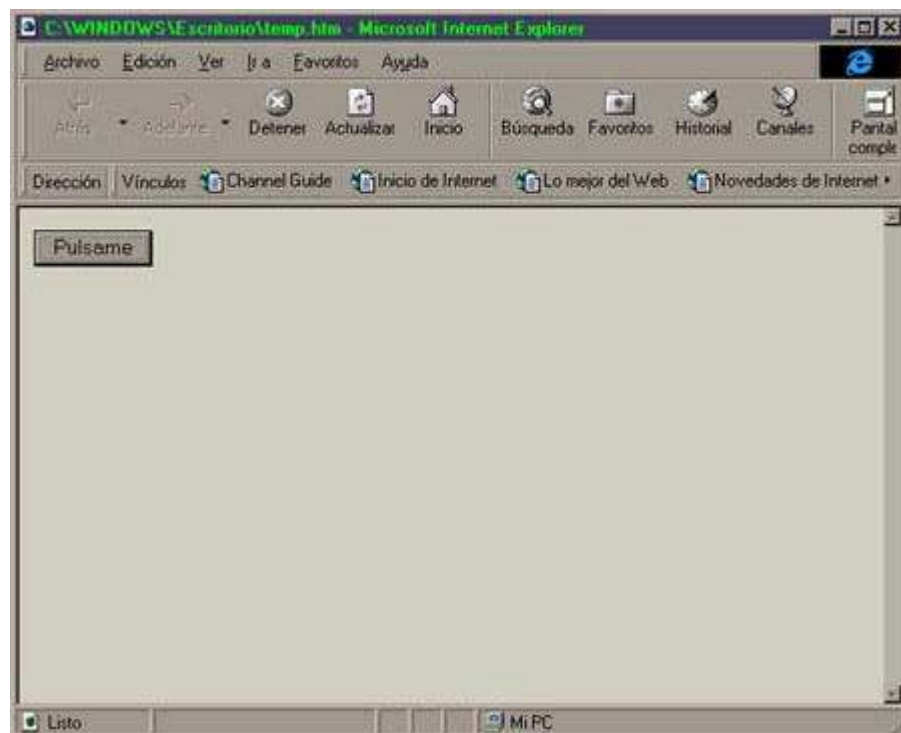
⁵ Es posible conseguirlo gratuitamente en <http://www.eu.microsoft.com/ie/download>

2. Primer programa

Vamos a realizar nuestro primer "programa" en JavaScript. Haremos surgir una ventana que nos muestre el ya famoso mensaje "Hola a todos". Así podremos ver los elementos principales del lenguaje. El siguiente código es una página Web completa con un botón que, al pulsarlo, muestra el mensaje.

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function Hola() {
    alert("Hola a todos");
}
</SCRIPT>
</HEAD>
<BODY>
<FORM>
<INPUT      TYPE="button"      NAME="boton"      VALUE="Pulsame"
onClick="Hola()" ">
</FORM>
</BODY>
</HTML>
```

Este es nuestro ejemplo en una web:



Al hacer clic sobre el botón que podemos ver en esta web, aparece esta ventana:



Ahora vamos a ver, paso por paso, qué significa cada uno de los elementos extraños que tiene la página anterior:

```
<SCRIPT LANGUAGE="JavaScript">
</SCRIPT>
```

Dentro de estos elementos será donde se puedan poner funciones en JavaScript. Puedes poner cuantos quieras a lo largo del documento y en el lugar que más te guste. Yo he elegido la cabecera para hacer más legible la parte HTML de la página. Si un navegador no acepta JavaScript no leerá lo que hay entremedias de estos elementos. Así que si programamos algo que sólo funcione con la versión 1.1 pondríamos LANGUAGE="JavaScript1.1" para que los navegadores antiguos no lean el código.

```
function Hola() {
    alert("Hola a todos");
}
```

Esta es nuestra primera función en JavaScript. Aunque JavaScript esté orientado a objetos no es de ningún modo tan estricto como Java, donde nada está fuera de un objeto. Para las cosas que se van a hacer en este curso, no vamos a crear ninguno, pero usaremos los que vienen en la descripción del lenguaje. En el código de la función vemos una llamada al método *alert* (que pertenece al objeto *window*) que es la que se encarga de mostrar el mensaje en pantalla. Por un fallo del Netscape no se pueden poner las etiquetas HTML de caracteres especiales en una función: no los reconoce.

```
<FORM>
<INPUT      TYPE="button"      NAME="boton"      VALUE="Pulsame"
onClick="Hola()" ">
</FORM>
```

Dentro del elemento que usamos para mostrar un botón vemos una cosa nueva: *onClick*. Es un evento. Cuando el usuario pulsa el botón, el evento *onClick* se dispara y ejecuta el código que tenga entre comillas, en este caso la llamada a la función *Hola()*, que tendremos que haber definido con anterioridad. Existen muchos más eventos que iremos descubriendo según avancemos en el curso.

En realidad, podríamos haber escrito lo siguiente:

```
<FORM>
```

```
<INPUT          TYPE="button"          NAME="boton"          VALUE="Pulsame"
onClick="alert('Hola a todos')">
</FORM>
```

y nos habríamos ahorrado el tener que escribir la función y todo lo que le acompaña, además de conseguir que nos reconozca el carácter especial que queramos.

3. Sintaxis

La sintaxis de JavaScript es como la de Java y, por lo tanto, muy parecida a C++. Las instrucciones terminan con un punto y coma y se agrupan mediante llaves; una doble barra (//) indica que el resto de la línea es un comentario. Los operadores matemáticos también son los mismos: + (que también sirve para cadenas), -, *, /, % (módulo), ++ y --.

Las asignaciones (=) son también como en C y C++, incluyendo +=, -=, *=, /= y %=. En cuanto a los operadores lógicos tenemos ||, &&, !, ^(xor), << y >> y las comparaciones son <, >, <=, >=, == y !=. Todo como en C.

Variables

Si la declaración de una variable se hace dentro de una función, dicha variable es local, en caso contrario, es global. En JavaScript no se asigna en la declaración el tipo de la variable sino que el intérprete se encarga de ello. Por ejemplo,

```
Var soyunavariabile;
Soyunavariabile=2;
```

Declararía una variable de un tipo entero. También se pueden crear objetos de la siguiente manera:

```
Var MiPrimerObjeto = new Object ();
MiPrimerObjeto.Colonia = "CKOne";
MiPrimerObjeto.TutorDeJavaScript = "Paco";
```

También, y como caso particular de objeto, podemos declarar arrays:

```
Var UnArray = new Array ();
UnArray[1]="Soy el primer elemento";
UnArray[2]="Los últimos serán los primeros";
```

4. Estructuras de control

Vamos a repasar por encima las estructuras de control existentes en JavaScript. Primero examinaremos las sentencias de salto. La

secuencia if...else nos permite realizar una bifurcación dependiendo del resultado de una condición lógica:

```
if (numero==1) {  
    numero++;  
    numero+=2; }  
else  
    numero--;
```

Una estructura similar y muy típica en C es la siguiente:

```
numero = (valor>=4) ? 3 : valor;
```

que, dependiendo del resultado de la condición expresada entre el `=` y la `?` nos permite dar a la variable *numero* un 3 si la condición resultase verdadera y *valor* si resultara ser falsa. La estructura *switch* nos permite bifurcar dependiendo del valor de una variable (que en el ejemplo sería *numero*):

```
switch (numero) {  
    case 1: alert ('Adios');  
    break;                                     // si no ponemos break, ejecutaría  
                                              // alert('Hola') y todas las  
                                              // instrucciones hasta encontrar un  
                                              // break o el final del switch  
  
    case 2:  
    alert ('Hola');  
    break;  
  
    default:                                   // si no es ninguno de los anteriores  
    alert ('No te entiendo');
```

Para hacer bucles podremos utilizar las siguientes estructuras: *for*, *for in*, *while* y *do/while*. La estructura *for(inicio;final;incremento)* nos permite ejecutar el contenido del bucle mientras la condición de *final* no se cumpla. Al comenzar la ejecución del bucle se ejecutará la sentencia *inicio* y en cada iteración *incremento*. La manera más habitual de usar estas posibilidades es la siguiente:

```
var factorial=1;  
for (n=2;n<=numero;n++) {  
    factorial+= factorial*n  
};
```

La estructura *for...in* es una novedad incluida en JavaScript y nos sirve para recorrer todos los elementos de un array:

```
for (contador in UnArray)  
{  
    UnArray[contador]="soy el elemento"  
};
```

Por último, las estructuras *while* y *do/while* no permiten recorrer un bucle mientras se cumpla una condición. La diferencia entre ellas es

que la primera comprueba dicha condición antes de realizar una iteración y la segunda lo hace después:

```
Var numero0;  
While (numero==1) {  
    Alert ('soy un while');  
}  
do {alert('soy un do/while');  
} while (numero==1);
```

En este caso sólo veríamos aparecer una ventana diciendo que es un do/while. La razón es evidente. El while comprueba primero si numero es igual a 1 y, como no lo es, no ejecutaría el código que tiene dentro del bucle. En cambio, el do/while primero ejecuta el bucle y luego, viendo que la condición es falsa, saldría.

5. Objetos

En el lenguaje JavaScript existe una serie de objetos predefinidos que se refieren a cosas como la ventana actual, el documento sobre el que trabajamos, o el navegador que estamos utilizando. Vamos a hacer un pequeñísimo repaso de algunos de ellos con los métodos y propiedades más usados.

5.1. Objeto window

Es el objeto principal. Define la ventana sobre la que estamos trabajando y tienen como descendientes los objetos referentes a la barra de tareas, el documento o la secuencia de direcciones de la última sesión. Ahora veremos sus métodos y propiedades más usadas.

open

```
[window.]open ("URL", "nombre", "propiedades");
```

El método open sirve para crear (y abrir) una nueva ventana. Si queremos tener acceso a ella desde la ventana donde la creamos deberemos asignarle una variable, si no simplemente invocamos el método: el navegador automáticamente sabrá que pertenece al objeto window. El parámetro URL contendrá la dirección de la ventana que estamos abriendo: si está en blanco, la ventana se abrirá con una página en blanco. El nombre será el que queramos que se utilice como parámetro de un TARGET y las propiedades son una lista separada por comas de algunos de los siguientes elementos:

```
Toolbar=yes/no  
Location=yes/no  
Directories=yes/no  
Status=yes/no  
Menubar=yes/no
```

```
Scrollbars=yes/no  
Resizable=yes/no  
Width=pixels  
Height=pixels
```

close

```
Variable.close();
```

Cierra la ventana Variable.

alert

```
Variable.alert("mensaje");
```

Muestra una ventana de diálogo en la ventana Variable con el mensaje especificado.

status

Define la cadena de caracteres que saldrá en la barra de estado en un momento dado.

defaultStatus

Define la cadena de caracteres que saldrá por defecto en la barra de estado. Cuando no la especificamos, defaultStatus será igual al último valor que tomó status.

5.2. Objeto document

Es un objeto derivado de window que identifica a un documento HTML.

write

```
document.write("cadena");
```

Escribe en un documento HTML, en el lugar del mismo donde hayamos situado el script que contiene al método, la cadena dada. El método writeln hace lo mismo, pero incluyendo al final un retorno de carro.

lastModified

Contiene la fecha y hora en que se modificó el documento por última vez y se suele usar en conjunción con write para añadir al final del documento estas características.

bgColor

Modifica el color de fondo del documento. El color deberá estar en el formato usado en HTML. Es decir, puede ser red o FF0000 (por ejemplo).

5.3. Objeto history

Este objeto se deriva de document y contiene todas las direcciones que se han visitado en la sesión actual. Tiene estos tres métodos:

document.history.back(): Volver a la página anterior.

document.history.forward(): Ir a la página siguiente,

document.history.go(donde): Ir a donde se indique, siendo donde un número tal que go(1)=forward() y go(-1)=back.

5.4. Objeto history

Este objeto derivado de document se refiere a un formulario. Puede ser útil para verificarlos antes de enviarlos.

Submit

Nombre.submit()

Envía el formulario llamado Nombre.

Text

Contiene el texto contenido en un campo de edición de un formulario.

5.5. Objeto date

Este es un objeto de propósito general que permite trabajar con fechas y horas.

5.6. Objeto Math

Este otro objeto de propósito general incluye las diversas funciones matemáticas más comunes, como abs o sin o el valor PI.

5.7. Objeto Navigator

A través de este objeto podremos averiguar varias características del navegador que usamos. Por ejemplo:

navigator.appName: Nombre del navegador.
navigator.appVer: Número principal de versión.
navigator.language: Idioma del mismo. (Versión JavaScript 1.2)
navigator.platform: Plataforma donde está ejecutándose. (Versión JavaScript 1.2)

No podemos sobrescribir estos atributos, pero sí leerlos.

5.8. Objeto Screen (Versión JavaScript 1.2)

Con este objeto podremos averiguar la configuración de la pantalla. Al igual que el anterior, todos sus atributos son de sólo lectura.

screen.height: Altura.
screen.width: Anchura.
screen.pixelDepth: Número de bits por pixel.

6. Eventos

Un evento, como su mismo nombre indica, es algo que ocurre. Para que una rutina nuestra se ejecute sólo cuando suceda algo extraño deberemos llamarla desde un controlador de eventos. Estos controladores se asocian a un elemento HTML y se incluyen así:

```
<A HREF= "http://home.netscape.com" onMouseOver="MiFuncion()" ">
```

6.1. Lista de eventos

Aquí tenemos una pequeña lista de eventos definidos en JavaScript.

Evento	Descripción	Elementos que lo admiten
OnLoad	Terminar de cargarse una página.	<BODY ...>
OnUnLoad	Salir de una página (descargarla).	<BODY ...>
OnMouseOver	Pasar el ratón por encima.	 <AREA ...>
OnMouseOut	Que el ratón deje de estar encima	 <AREA ...>
OnSubmit	Enviar un formulario	<FORM ...>
OnClick	Pulsar un elemento	<INPUT TYPE="button,checkbox,radio" ...>
OnBlur	Perder el cursor	<INPUT TYPE="text" ...>

Evento	Descripción	Elementos que lo admiten
		<TEXTAREA ...>
OnChange	Cambiar de contenido o perder el cursor	<INPUT TYPE="text"> <TEXTAREA ...>
OnFocus	Conseguir el curso.	<INPUT TYPE="text" ...> <TEXTAREA ...>
OnSelect	Seleccionar texto.	<INPUT TYPE="text" ...> <TEXTAREA ...>

Un ejemplo

Como ejemplo, vamos a hacer que una ventana aparezca automáticamente en cuanto pasemos un cursor por encima de un elemento <A> (e impidiendo, de paso, que quien esté viendo la página pueda hacer uso del mismo).

```
<HTML>
<HEAD>
  <SCRIPT LANGUAGE="JavaScript">
    <!-- Se usan los comentarios para esconder
         el código a los navegadores sin JavaScript
    function Alarma() {
      alert ("No me pises, que llevo chancas");
      return true;
    }
    // -->
  </SCRIPT>
</HEAD>
<BODY>
<A HREF="eventos.html" onMouseOver="Alarma()" ">
Pasa por aquí
</A>
</BODY>
</HTML>
```

7. Trucos

7.1 Crear nuevas ventanas

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
  function AbrirVentana () {
    MiVentana=open("", "Mi
    ventana", "toolbar=no, directories=no, menubar=no");
    MiVentana.document.write("<HTML><HEAD><TITLE>Una          nueva
    ventana</TITLE></HEAD>");
    MiVentana.document.write("<BODY><CENTER><H1><B>Esto puede ser lo
    que tu quieras</B></H1></CENTER></BODY></HTML>");
  }
</SCRIPT>
</HEAD>
<BODY>
```



```
<FORM>
<INPUT      TYPE="button"      NAME="boton1"      VALUE="Abrete,      Sésamo"
onClick="AbrirVentana()" ">
</FORM>
</BODY>
</HTML>
```

Este ejemplo muestra la posibilidad de abrir nuevas ventanas de nuestro navegador con JavaScript. Se llama a la función AbrirVentana desde el evento onClick, como ya sabemos hacer. Esta función crea la nueva ventana MiVentana y escribe en ella por medio del objeto document todo el código HTML de la página.

7.2. La barra de estado

Si te has fijado en las webs que hayas visitado, habrás notado que algunas gobiernan sobre la barra de estado del navegador.

Vamos a hacer lo más sencillo: escribir mensajes diversos en la barra de estado. Hablábamos sobre los objetos predefinidos y aparecía el objeto window. En este objeto se definían dos atributos: status y defaultStatus. Para poner un mensaje en la barra de estado nada más cargar el documento y que se mantenga ahí mientras no haya otro más importante (un sustituto del famoso Document: Done del Netscape) haremos:

```
<BODY onLoad>window.defaultStatus='El document ya se ha leído';return true">
```

El código lo único que hace es modificar *defaultStatus* y devolver *true* como resultado del controlador de eventos. Por alguna misteriosa razón es obligatorio hacer esto cuando modificas algo de la barra de estado. Al parecer no es más que una convención.

Ahora veremos como se puede definir el valor de la barra de estado cuando el ratón pasa por encima de un elemento <A>:

```
<A      HREF="mipagina.html"      onMouseOver="window.status='Vamos      a      mi
pagina';return true">mipagina</A>
```

Vamos a ver como se hacen scrolls, aunque personalmente no os recomiendo que los utilicéis ya que suelen confundir a los usuarios (no vemos los mensajes del browser) y despistan bastante.

Colocamos el siguiente código:

```
<HTML>
<HEAD>
<TITLE>Ejemplo scroll</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var text="      Este es el mensaje que vamos a ver moverse";
var longitud=texto.length;
function scroll() {
text=texto.substring(1,longitud-1)+texto.charAt();
```

```
window.status=texto;
setTimeout("scroll()",150);
}
</SCRIPT>
</HEAD>
<BODY onLoad="Scroll(); return true;">
Esta página contiene un scroll
</BODY>
</HTML>
```

Como podéis ver, la cosa no es ni más larga ni más compleja que los ejemplos anteriores. Simplemente escribe el texto en la barra de estado y luego coge el carácter más a la izquierda del mismo y lo pone a la derecha, para después volver a escribirlo. La única pega que tiene es cómo hago para que la función se llame cada cierto tiempo predeterminado para ir desplazando el texto a una velocidad constante.

La respuesta está en el método setTimeout("función", tiempo) que llama a función cuando hallan pasado tiempo milisegundos.

7.3. Volver a la página anterior

Si quieres volver a la última página donde has estado, o a cinco más atrás, lo puedes hacer con los métodos proporcionados por document.history que ya vimos. Para emplearlos no tienes más que hacer lo siguiente:

```
<A HREF="javascript:window.history.back()">Volver atrás</A>
```

7.4. Tratamiento de imágenes (disponible en versiones 1.1. y posteriores)

Los que vamos a lograr con este truco es un pequeño cambio de imágenes. Al pasar el ratón por encima de una opción podrás observar que el gráfico cambiar. Para hacer esto deberemos crear dos gráficos distintos: el que se verá normalmente y el que únicamente podrá verse cuando el ratón pase por encima. Si llamamos al primero, por ejemplo, apagado.gif y al segundo encendido.gif el código necesario para que el truco funcione es:

```
<HTML>
<HEAD>
<TITLE>Ejemplo de tratamiento de imágenes</TITLE>
<SCRIPT LANGUAGE="JavaScript1.1">
var activado=new Image();
activado.src="encendido.gif";
var desactivado= new Image();
desactivado.src="apagado.gif";
function activar(nombreImagen) {
document.[nombreImagen].src=desactivado.src;
}
</SCRIPT>
```

```
</HEAD>
<BODY>
<A      HREF="mipagina.html"      onMouseOver="activar('prueba');"
onMouseOut="desactivar('prueba');">
<IMG NAME="prueba" SRC="apagado.gif" BORDER=0>
</A>
</BODY>
</HTML>
```

Lo primero que hay que indicar es que para que funcione el invento la imagen debe ser un enlace. ¿Por qué? Porque los eventos que controlan si el ratón pasa o no por encima no son admitidos por la etiqueta ****. También deberemos "bautizar" nuestra imagen usando la etiqueta **NAME="como-se-llame"** para permitir su identificación posterior.

El ejemplo funciona de la siguiente manera: en principio la imagen que vemos es la desactivada, que es la que indica el elemento ****. Al pasar el ratón por encima de ella el evento **onMouseOver** llamará a la función **activar** llevando como parámetro el nombre de la imagen. Esta función sustituirá en el objeto **document** el nombre del fichero donde se guarda la imagen por **encendido.gif**, que es el gráfico activado. Cuando apartemos el ratón de la imagen, será el evento **onMouseOut** el que se active, llamando a **desactivar**. Esta función sustituirá el gráfico de nuevo, esta vez por **apagado.gif**.

Leyendo esta explicación parece que una parte del código sobra. ¿Para qué sirve declarar dos objetos **image** para albergar los gráficos? ¿No bastaría con cambiar directamente el nombre de la imagen escribiendo **document.[nombreImagen].src='encendido.gif'**? Pues no del todo. Efectivamente funcionaría, pero cada vez que cambiásemos el nombre el navegador traería la imagen del remoto lugar donde se encontrara. Es decir, cada vez que pasásemos el ratón por encima de la imagen o nos alejásemos de ella tendríamos que cargar (aunque ya lo hubiésemos hecho antes) el gráfico correspondiente. Es cierto que con la caché del navegador este efecto quedaría algo mitigado, pero siempre es mejor precargar las imágenes usando el objeto **Image**, ya que así el navegador comenzará a leer el gráfico cuando ejecute el objeto **Image**, ya que así el navegador comenzará a leer el gráfico en cuanto ejecute el código en vez de esperar a que el usuario pase por encima de la imagen con el ratón. El objeto **Image** tiene como atributos los distintos parámetros que puede tener la etiqueta ****.

Este código tiene dos defectos bastante graves. En los navegadores que no poseen JavaScript 1.1 (léase Netscape 2 y Explorer 3 e inferiores) las funciones para activar y desactivar la imagen no estarán definidas y el navegador dará un error al no poder encontrarlas. La solución a este problema consiste en definir las para cualquier navegador con **<SCRIPT LANGUAGE="JavaScript">** y detectar el mismo usando el objeto *navigator*. Así podremos no crear las variables que guardan los gráficos ni ejecutar el código de las funciones para activar y

desactivar en el caso de que el navegador no soporte esta nueva versión de JavaScript.

El segundo defecto es debido a la insistencia de Microsoft en hacer su Jscript incompatible con el JavaScript de Netscape. Los gráficos en Jscript no se identifican por su nombre sino por su número de orden dentro de la página. Es decir, el primer gráfico que aparezca en el documento HTML será el número 0, el siguiente el 1, etc... Eso sí sólo estarán numeradas aquellas imágenes que tengan nombre. ¿Cómo solucionar esto de una manera elegante, programando una función que nos sirva para los dos? Llamando a nuestros gráficos con el número que les corresponda. El código ya sin defectos es el siguiente:

```
<HTML>
<HEAD>
<TITLE>Ejemplo de imágenes</TITLE>
<SCRIPT LANGUAGE="JavaScript">
navegador = navigator.appname;
versionNav = parseInt(navigator.appversion);
if ((navegador == "Netscape" && versionNav >= 3) ||
    (navegador.indexOf("Explorer") && versionNav >= 4))
    version = 3;
else version = 2;

if (version == 3) {
var activado=new image();
activado.src="encendido.gif";
var desactivado = new image ();
desactivado.src="apagado.gif";
}
function activar(nombreimagen) {
    if (version == 3) {
        document[eval(nombreImagen)].src=activado.src; }
}
function desactivar(nombreImagen) {
    if (version == 3) {
        document[eval(nombreImagen)].src=desactivado.src; }
}
</SCRIPT>
</HEAD>
<BODY>
<IMG SRC="deberiaser0.gif" BORDER=0>
<IMG SRC="deberiaser1.gif" BORDER=0>
<A      HREF="mipagina.html"      onMouseOver="activar('0'); "
moMouseOut="desactivar('0'); ">
<IMG NAME="0" SRC="apagado.gif" BORDER=0>
</A>
</BODY>
</HTML>
```

Al aplicar la función eval, Netscape, que espera una cadena de caracteres, recibirá una cadena de caracteres. El Explorer, que espera un número entero, recibirá ese número.

8. Cookies

Para más que necesario definir algo que lleva el absurdo nombre de galletita o *cookie*. Una galleta es un elemento de una lista que se guarda en el fichero *cookies.txt* en el ordenador del visitante. Cada elemento de esa lista tiene dos campos obligatorios: el nombre y su valor; y uno opcional: la fecha de caducidad. Este último campo sirve para establecer la fecha en la que se borrará la galleta. Tiene este formato:

```
Nombre=valor; [expires=caducidad]
```

Sólo el servidor que ha enviado al usuario una determinada *cookie* puede consultarla. Cada galleta tiene un tamaño máximo de 4 Kb y puede haber un máximo de 300 cookies en el disco duro. Cada servidor podrá almacenar como mucho 20 galletas en el fichero *cookies.txt* (en el caso de usar Netscape) o en el directorio *cookies* (si utilizamos Explorer) del usuario. Si no especificamos la fecha de caducidad de la galleta la galleta se borrará del disco duro del usuario en cuanto éste cierre el navegador.

8.1. Funciones básicas

Para poder hacer algo con cookies deberemos programar dos funciones: una que se encargue de mandar una galleta al usuario y otra que consulte su contenido.

```
Function mandarGalleta(nombre,valor,caducidad) {  
Document.cookie = nombre + "=" + escape(valor)) + ((caducidad==null) ?  
"" : ("; expires=" + caducidad.toGMTString()))  
}
```

Con esta función mandamos una galleta. Vemos que el valor es codificado por medio de la función *escape* y que la caducidad (en caso de decidir ponerla) debe ser convertida a formato GMT. Esto se hace mediante el método *toGMTString()* del objeto *Date*.

```
function consultarGalleta () {  
var buscamos = nombre + "=";  
if (document.cookie.length > 0) {  
    i = document.cookie.indexOf(buscamos);  
    if (i != -1) {  
        i += buscamos.length;  
        j = document.cookie.indexOf(";", i);  
        if (j == -1)  
            j = document.cookie.length;  
        return unescape(document.cookie.substring(i, j));  
    }  
}  
}
```

Declaramos la variable, buscamos que contiene el nombre de la galleta que queremos buscar más el igual que se escribe justo después de cada nombre, para que así no encontremos por error un valor o una subcadena de otro nombre que sea igual al nombre, para que así no encontremos por error un valor o una subcadena de otro nombre que sea igual al nombre de la galleta que buscamos. Una vez encontrada extraemos la subcadena que hay entre el igual que separa el nombre y el valor y el punto y coma con que termina dicho valor.

8.2. Un ejemplo

Vamos a ver un ejemplo. Utilizaremos una galleta llamada VisitasANuestraWeb para guardar el número de veces que se ha visitado nuestra web:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
<!-- Se usan los comentarios para esconder el código a navegadores sin
JavaScript
function Contador () {
    var fecha = new Date(99,12,31); // La fecha de caducidad es
    31-12-1999
    if !(num=consultarGalleta("VisitasANuestraWeb")) num=0;
    num=0;
    num++;
    mandarGalleta("VisitasANuestraWeb",num,fecha);
    if (num==1) document.write("esta es la primera vez que
    entras.");
    else {
        document.write("has visitado esta web "+num+"
        veces.");
    }
}
//→
</SCRIPT>
</HEAD>
<BODY>
Por lo que veo,
<SCRIPT LANGUAGE="JavaScript">
<!--
Contador()
//→
</SCRIPT>
</BODY>
</HTML>
```

9. Futuro

JavaScript, como todas las (en este caso, relativamente recientes) tecnologías de Internet evoluciona a un ritmo muy rápido, por lo que podemos decir que siempre aparecen nuevos avances, por lo que conviene estar siempre alerta suscribiéndonos a listas de distribución,

grupos de noticias, etc que incluyan nuestros temas de interés tanto en JavaScript como en cualquier tema relacionado con el web.

Ahora veremos algunas de las innovaciones introducidas en JavaScript.

9.1. Ficheros .js (Versiones JavaScript 1.1 y superiores)

Un fichero .js es un archivo donde podremos guardar funciones y variables globales que podrán leerse desde cualquier página HTML. Gracias a ellos podremos evitar tener que duplicar funciones que se necesiten en más de un documento. Podremos incorporarlos a nuestras páginas de esta manera:

```
<HTML>  
<HEAD>  
<TITLE>Mi página</TITLE>  
<SCRIPT SRC="funciones.js">  
<!--  
alert('Error en fichero js');  
//-->  
</SCRIPT>  
</HEAD>  
<BODY>  
Lo que sea.  
</BODY>  
</HTML>
```

El código que incluyamos entre un <SCRIPT SRC> y un </SCRIPT> sólo se ejecutará en caso de que la lectura del fichero .js falle.

En esta versión 1.1 se define también el método para incluir una expresión JavaScript en los parámetros de una etiqueta HTML. La forma de hacerlo es considerando dicha expresión (entre llaves) como un carácter especial de HTML, encerrándolo por tanto, entre un & y un ;. Por ejemplo:

```
<HR WIDTH=&{"&";{anchoLinea+"%";}};>
```

que, en el supuesto de que la variable anchoLinea sea igual a 50 nos daría una línea HR de ancho 50%.

9.2. HTML Dinámico (Versiones JavaScript 1.2 y superiores)

El JavaScript 1.2 trae unas cuantas cosas nuevas, pero la mayoría tienen que ver con una nueva especificación llamada HTML dinámico. Es un tema demasiado amplio como para tratarlo aquí,

necesitaría un curso para él sólo. Así que me voy a limitar a comentar por encima las dos características del HTML dinámico que tienen que ver con JavaScript: las hojas de estilo y las capas.

Los que hayan hecho páginas web para Internet Explorer 3 seguramente ya saben qué son y como funcionan las hojas de estilo. En resumen, estas hojas intentan separar el contenido de una página de la forma de presentarlo en pantalla. La novedad es que ahora podremos definir dichas hojas de dos maneras: en cascada (el modo tradicional del Explorer) o con JavaScript. Si queremos, por ejemplo, que todos los párrafos estén en rojo haremos:

En cascada	JavaScript
<code><STYLE TYPE="text/css"> P {COLOR: red;} </STYLE></code>	<code><STYLE TYPE="text/javascript"> tags.P.color="red"; </STYLE></code>

Las capas son un concepto nuevo introducido con el Communicator. Se pueden definir como páginas HTML que se insertan dentro de otra página. Sin JavaScript no tendrían mucha utilidad, pero con la mezcla de los dos podremos realizar animaciones, páginas que cambien según deseemos, etc...

Entre las etiquetas `<LAYER>` y `</LAYER>` podemos incluir todo lo que queramos: todo lo que podemos meter en el cuerpo de un documento HTML. Estas etiquetas tienen entre sus parámetros cosas como la posición en la pantalla, si están ocultas o visibles o si son opacas y no permiten ver las posibles capas que tengan debajo. Todos estos atributos son modificables usando JavaScript.

Parece sencillo pensar que, si en una capa metemos un gráfico cualquiera, podemos hacer uso del JavaScript y la función `setTimeout` para modificar sus posiciones y así lograr una animación. Del mismo modo, si modificamos los atributos de visibilidad desde un evento `onSubmit` de un formulario podremos conseguir que el usuario elija qué parte de nuestra página web quiere ver.

D. Java

1. ¿Qué es Java?
2. Una breve historia de Java
3. Computación distribuida
4. Objetos, objetos, objetos
5. La solución independiente de la plataforma
6. Un binario para todos
7. La sintaxis HTML para los applets de Java
 - 7.1. El tag <APPLET>
 - 7.2. El tag <PARAM>
 - 7.3. Otros elementos HTML
8. Aspectos de diseño
 - 8.1. Estética de diseño
 - 8.2. Color
 - 8.3. Apariencia
 - 8.4. Manía por la animación
 - 8.5. Diseño funcional
 - 8.6. Tamaño de los applets
 - 8.7. Cómo colocar los applets
 - 8.8. Sonido
 - 8.9. Evite atiborrarse de applets
 - 8.10. Tiempo de transferencia

D. Java

1. ¿Qué es Java?

Java es tal vez uno de los avances en la tecnología de Internet sobre los que más se ha hablado desde World Wide Web. Personas de todas las áreas de la computación están discutiendo acerca de Java y de la manera en la que cambiará el panorama de la computación. Sin duda alguna, Java cambiará (de hecho, ya lo ha hecho) la manera en la que la gente usa Internet y las aplicaciones en red. Java ha introducido muchas ideas al excenario de World Wide Web y estas ideas representan varios cambios interesantes en Internet y la computación en general.

Debido a la parafernalia que rodea a Java, Java se ha llegado a convertir en algo esotérico. En el corazón de este esoterismo se encuentra Java. Java es un lenguaje de programación orientado a objetos (OOP o LPOO) que emplea muchos elementos comunes de otros lenguajes LPOO como por ejemplo C++, pero añade algunas mejoras que facilitan la programación. Al igual que cualquier otro lenguaje, Java cuenta con una sintaxis particular, una estructura para programas y muchas aplicaciones de soporte.

El Java Developer's Kit (JDK) contiene todas las herramientas necesarias para crear aplicaciones (o applets web) mediante el lenguaje de programación Java, incluyendo las siguientes:

Javac	El compilador de Java.
Jdb	El depurador de Java.
Javadoc	El programa de documentación de Java.
Java	La Máquina Virtual de Java (Java Virtual Machine)
Appletviewer	El visor de applets de Java.

Tal vez algunos de estos componentes de Java no parezcan muy obvios, como la Máquina Virtual de Java. Al usar Java para programar, uno no accede a la Máquina Virtual de Java directamente. Sin embargo, la usan otros visualizadores (browsers) comerciales de Web que son capaces de ejecutar Java, así como appletviewer. Todos estos elementos se relacionan o vinculan directamente con Java y lo emplean en World Wide Web.

2. Una breve historia de Java

Java vio la luz en 1990 como un nuevo lenguaje llamado Oak. Sun Microsystems⁶ había establecido un grupo de proyecto cuyo nombre en

⁶ <http://www.sun.com>

clave era *green* (verde) para desarrollar productos nuevos y expandir los mercados de Sun. Oak fue desarrollado originalmente para un asistente personal digital llamado *7 que Sun pretendía lanzar al mercado con una interfaz gráfica para usuario de acción continua.

El *7 nunca fue lanzado al mercado y con el tiempo Sun formó una compañía llamada FirstPerson para desarrollar el *7 en dispositivos especiales para la televisión interactiva. Debido a una variedad de circunstancias, la promesa de la televisión interactiva pronto se desvaneció y Oak se quedó sin mercado. Sin embargo, aproximadamente por las mismas fechas que FirstPerson y Oak fallaban, World Wide Web iniciaba su despegue. Compañías como Netscape comenzaron a crear software que llevó a WWW a ser el centro de atención en Internet. Sun pronto se dio cuenta de que Oak tenía posibilidades en Web y poco después Oak apareció en Internet con un nuevo nombre: Java.

Java se encuentra en su primera versión como ambiente de desarrollo y ya está empezando a influir en la dirección de la computación e Internet. El lenguaje de programación Java se puede adquirir sin costo alguno a través de Internet y Sun está otorgando licencias de estructuración completa de Java⁷ y sus componentes a diversos fabricantes de software para Internet con la esperanza de crear un nuevo estándar de programación para Web.

3. Computación distribuida

Java es revolucionario por varias razones, pero una de las más importantes es cómo cambia la manera en que usamos nuestras computadoras. Java está diseñado alrededor del concepto de las redes y la conectividad. En el mundo de la computación actual ejecutamos las aplicaciones desde el disco duro de la computadora; instalamos aplicaciones, las usamos y las actualizamos cuando se vuelven obsoletas. Si necesitamos ponernos en contacto con alguien desde nuestra máquina, arrancamos una aplicación, como un visualizador Web o un software de correo electrónico, que se conecta a una red. Pero imagina que en lugar de esto tus aplicaciones estuvieran en un servidor de red y tu máquina las descargara cada vez que las necesitas. En este modelo, tu software siempre sería actual y tu máquina siempre estaría en contacto con la red. La computación distribuida se refiere a compartir recursos de programas a través de redes.

La finalidad de Java es producir aplicaciones muy completas para red. Sin embargo, el ancho de banda y las restricciones de red actuales ocasionan que esta realidad sea todavía distante. En lugar de ello, Java se utiliza para agregar funcionalidad y diversión a aplicaciones de red tales como los visualizadores Web. Con Java puedes escribir

⁷ <http://java.sun.com>

miniaplicaciones, llamadas *applets*, que pueden incorporarse dentro de sitios web y ser ejecutadas desde una página base. Por ejemplo, el applet tickertape desplaza el texto a través de la pantalla de una página web.

El corazón de las aplicaciones y applets distribuidos es la comunicación por red. Java está diseñado para usarse con Internet y otras redes de gran escala, incluye un gran número de bibliotecas de red para utilizar redes TCP/IP y varios otros protocolos que se emplean comunmente en Internet, como HTTP y FTP. Esta característica significa que puedes integrar con facilidad servicios de redes dentro de sus aplicaciones. El resultado final es una amplia gama de funcionalidad en la red y aplicaciones que se prestan para su uso en red.

El concepto de applet también encaja bien dentro de un modelo de software distribuido. Un applet es en esencia una aplicación reducida. Los ejecutables de las aplicaciones pueden ser bastante grandes y con frecuencia las aplicaciones cuentan con una serie de archivos de soporte y configuración que necesitan para poder ejecutarse. Todos estos archivos dan como resultado programas muy grandes que son difíciles de distribuir a través de una red. Los archivos de los applets son pequeños y transportables, lo cual permite que se puedan cargar e integrar con facilidad dentro del diseño de una página web. Los applets pueden agregarse a las páginas para dotarlas de nuevos niveles de funcionalidad, interactividad o simplemente algunos gráficos o sonidos que contribuyen a que un sitio web sea más interesante. Puesto que por lo general los applets son pequeños, no incrementan de manera significativa el tiempo de transferencia de una página y puesto que son programas reales, le ofrecen más flexibilidad que los scripts HTML y CGI tradicionales.

Java y los aspectos de seguridad

Java incluye características de seguridad para reforzar su empleo en Internet. Un problema de seguridad potencial tiene que ver con los applets de Java, que son código ejecutable que opera en nuestra máquina local. Java emplea verificación de código y acceso limitado al sistema de archivos para asegurarse de que el código no dañe nada en nuestra máquina local.

Cuando descarga un applet a su máquina local, el código es verificado para asegurarse de que el applet sólo emplee llamadas y métodos de sistema válidos. Si el applet pasa la verificación, se le permite ser ejecutado, pero a los applets no se les da acceso al sistema de archivos de la máquina local. Las restricciones de acceso al sistema de archivos limitan de cierta manera a los applets, pero también evitan que éstos borren accidentalmente archivos, introduzcan virus o causen cualquier tipo de desorden en nuestro sistema

4. Objetos, objetos, objetos

La programación orientada a objetos se ha ido introduciendo constantemente en el mundo de la programación desde hace varios años. El concepto no es difícil de entender, los objetos son pequeñas porciones de código diseñadas para realizar una función específica. Luego, un programador puede combinar estos objetos para crear un programa determinado.

Por ejemplo, un programa simple puede contener un objeto de entrada, un objeto de procesamiento y un objeto de salida. Por ejemplo, en un applet de calculadora, el teclado puede considerarse como el objeto de entrada. Éste recibe la información del usuario y luego la pasa al objeto procesador, el cual realiza los cálculos. Luego el objeto procesador pasa el resultado a un objeto de salida que despliega el resultado para el usuario. Los objetos están diseñados para hacer que la programación sea más flexible.

Aunque el concepto básico de la programación orientada a objetos es muy simple, su puesta en práctica suele ser una tarea muy compleja. Se necesita una gran cantidad de tiempo en la etapa de planteamiento del desarrollo de un applet para asegurar que los componentes están diseñados para ser transportables y funcionar juntos. A la larga, los objetos bien diseñados pueden facilitar la programación, pero requieren una inversión considerable en el planteamiento y desarrollo originales. Sin embargo, la OOP ofrece esperanzas para el futuro conforme Java crezca.

5. Una solución independiente de la plataforma

Java ha sido modelado para ajustarse a muchos proyectos diferentes desde su creación a principios de los noventa. Ha pasado de ser un lenguaje de programación para asistentes personales digitales, a lenguaje de programación para dispositivos de televisión interactiva y por último a su más reciente encarnación como lenguaje de programación para Web. Esta transformación deberá darnos una idea de la flexibilidad y transportabilidad de Java; está diseñado para ser independiente de la máquina y funcionar dentro de diferentes sistemas operativos. La transportabilidad es una de las principales metas de Java. El código de Java está diseñado para ser independiente de la plataforma y cuenta con varias características diseñadas para ayudarle a alcanzar dicha meta.

Cuando escribimos una aplicación Java, estamos escribiendo un programa diseñado para ejecutarse en una computadora muy especial: la Máquina Virtual de Java. Esta máquina virtual es el primer paso hacia una solución independiente de la plataforma. Cuando desarrollas

software en un lenguaje como C++, por lo general programamos para una plataforma específica, como una máquina de Windows o macintosh. El lenguaje de programación usará una variedad de funciones que están diseñadas específicamente para el procesador empleado por la máquina para la cual programas. Debido a que el lenguaje emplea instrucciones específicas para una máquina, tienes que modificar el programa para un nuevo procesador si deseas ejecutarlo en otra máquina. Esta tarea puede consumir mucho tiempo y recursos.

El código Java no está escrito para ningún tipo de computadora física. En lugar de ello, está escrito para una computadora especial llamada Máquina Virtual, que en realidad es otra pieza de software. Entonces la máquina virtual interpreta y ejecuta el programa de Java que has escrito. La máquina virtual está programada para máquinas específicas, por lo que existe una Máquina Virtual Windows 95, una Máquina Virtual par Sun, etc. Incluso hay una copia de la Máquina Virtual integrada dentro de Netscape, lo que permite a este visualizador ejecutar programas en Java.

Al transportar la Máquina Virtual de una plataforma a otra, en lugar de los programas Java mismos, cualquier programa de Java puede usarse en cualquier máquina que ejecute una copia de la Máquina Virtual. Esta característica es la razón por la cual un mismo applet de Java puede ejecutarse tanto en estaciones de trabajo UNIX como en máquinas Windows. Sun ha puesto un esfuerzo especial para transportar la Máquina Virtual a casi todos los principales tipos de máquinas en el mercado. Al hacer esto, Sun garantiza la transportabilidad de Java. El beneficio para nosotros es la capacidad de escribir su código una vez y luego usarlo en muchas máquinas.

6. Un binario para todos

La Máquina Virtual requiere un código binario especial para ejecutar los programas de Java. Este código, llamado bytecode (código de byte), no contiene instrucciones relacionadas específicamente con una plataforma. Esto significa que si escribes un programa en una estación de trabajo Sun y lo compila, el compilador generará el mismo bytecode que una máquina que ejecute Windows 95. Este código es el segundo paso hacia un ambiente de desarrollo independiente de la plataforma. Imagina el tiempo y el dinero que se podría ahorrar si en nuestros días fuera posible escribir el software una sola vez y luego emplearlo en todas las computadoras. No habría necesidad de crear versiones del software para Mac o para Windows y no sería necesario contratar hordas de programadores para convertir algún programa de Mac a una versión para Windows.

Cuando compilamos un programa de Java, el compilador genera el bytecode independiente de la plataforma. Entonces, cualquier Máquina Virtual Java puede ejecutar dicho bytecode. Puede ser observado

mediante Netscape o appletviewer de Java, tanto en una estación de trabajo Sun o en una máquina Windows NT. Lo más importante es la transportabilidad. Un código binario se ajusta a todas las máquinas.

7. La sintaxis HTML para los applets de Java

Para crear el protocolo de comunicación entre HTML y Java se necesitan algunos comandos sencillos que explicaremos en los siguientes apartados.

7.1. El tag <APPLET>

Como ya se vio anteriormente, el tag HTML más importante en la configuración de Java es el tag <APPLET>, que sirve para vincular applets. Entre las marcas <APPLET> y </APPLET> pueden haber, junto a otros elementos HTML, tags <PARAM>, utilizados para transmitir los parámetros al applet.

Pero veamos primero los atributos del elemento APPLET. Este elemento posee once atributos, tres de los cuales son obligatorios.

CODE

Con el atributo CODE se define el nombre del archivo que contiene el código compilado del programa. El valor del atributo CODE es una variable de carácter. Por ejemplo: CODE="buenosdias.class". El atributo CODE debe indicarse obligatoriamente.

WIDTH

Dentro de la página que se está creando, al applet se le asigna un área rectangular que el browser administra prácticamente como si se tratara de una imagen corriente (de hecho en browsers de últimas generaciones el icono que aparece mientras se carga un programa Java es el de un gráfico). Con el atributo WIDTH se define el ancho de esta área en puntos de imagen. El valor del atributo WIDTH es un número entero y es obligatorio indicarlo.

HEIGHT

EL atributo HEIGHT, al que también se le debe asignar obligatoriamente un número entero como valor, indica la altura del área de pantalla que debe ocupar el applet. Esta altura se indica en puntos de pantalla.

CODEBASE

El atributo CODEBASE permite indicar el URL del directorio desde el cual se puede cargar el applet. Si este atributo no tiene asignado ningún valor, el código del applet se buscará en el mismo directorio desde el que se cargó la página HTML que contiene el tag <APPLET> correspondiente.

ALT

El atributo opcional ALT permite transmitir una variable de carácter que se representará en lugar del applet cuando se trabaje con un browser que no pueda representar un applet.

NAME

El valor del atributo NAME puede utilizarse para identificar el applet en su interacción con otros elementos activos. En este contexto resulta posible tanto la influencia mutua de dos applets como el control de un applet con ayuda de los lenguajes script. El atributo NAME es opcional.

ALIGN

El atributo ALIGN sirve para definir la posición relativa del applet dentro del documento HTML en que se representa. Este atributo es opcional y sus valores posibles se indican en la tabla siguiente:

Valor del atributo (ALIGN=)	Significado
Top	La alineación se produce dentro de una línea de texto de manera que el borde superior del applet coincide con el margen superior de la línea.
Middle	La alineación se produce dentro de una línea de texto de manera que el centro vertical del applet coincide con el centro de la línea.
Bottom	La alineación se produce dentro de una línea de texto de manera que el borde inferior del applet coincida con el borde inferior de la misma.
Left	El applet se alinea en el borde izquierdo del documento y queda rodeado con el texto.
Right	El applet se alinea en el borde derecho del documento y queda rodeado por el texto.
Center	El applet interrumpe la emisión del texto y queda centrado en una línea de documento exclusivamente para él.

HSPACE

La distancia entre un applet y el texto que le rodea se puede definir con el atributo opcional HSPACE. El valor de este atributo, en su caso, es siempre un número entero.

VSPACE

La distancia vertical entre un applet y el elemento HTML que le rodea se puede definir con el atributo VSPACE. Este atributo es opcional y su valor debe ser un número entero.

DOWNLOAD

El atributo **DOWNLOAD** sirve para definir el orden en que se deben cargar las imágenes del applet (si existen). Este atributo no forma parte de HTML 3.2. pero sí está contenido en el proyecto HTML "Cougar", la versión experimental del futuro estándar HTML. MSIE 3.0 soporta este atributo.

TITLE

Con el atributo **TITLE** se puede transmitir un título para el applet en forma de variable de carácter. Este atributo también se ha definido por primera vez en el proyecto HTML "Cougar" y ya ha sido implementado en MSIE 3.0

7.2. El tag PARAM

El tag HTML **<PARAM>** sólo está permitido dentro de los elementos **APPLET** y **OBJECT**. Sirve para transmitir el parámetro de objeto de programa definido en el tag inicial de estos elementos. El elemento **PARAM** consta exclusivamente de un tag inicial. Éste posee cuatro atributos que se describirán a continuación.

NAME

El atributo **NAME** especifica el nombre de la propiedad del objeto de programa correspondiente cuyo valor se ha de modificar. Este atributo es obligatorio.

VALUE

El valor del atributo **VALUE** es el mismo valor que se debe asignar a la propiedad de objeto indicada en el atributo **NAME**. La indicación de este atributo es siempre obligatoria.

TYPE

El atributo **TYPE** indica el "Internet-Media-Type". Este atributo es opcional y no está incluido todavía en la especificación HTML 3.2. La implementación en MSIE30 se realizó con motivo de la especificación en la versión experimental de HTML Cougar.

VALUETYPE

Con ayuda del atributo **VALUETYPE** se puede hacer una diferenciación en la interpretación del valor transmitido en el atributo **VALUE**. Los valores posibles de este atributo se representan en la tabla siguiente:

Valor del atributo (VALUETYPE=)	Significado
data	El valor es un dato. Este es el valor predefinido.
Ref	El valor es un URL.

Object	El valor es el URL de un objeto.
---------------	---

Tampoco el atributo **VALUE** forma parte de **HTML 3.2** pero fue incluido en **MSIE 3.0** como adelanto a una posible versión futura.

7.3. Otros elementos HTML

En el elemento **APPLET** pueden haber tantos elementos **HTML** como se desee, aunque éstos siempre se considerarán como posible alternativa al propio applet. En el ejemplo siguiente se ha previsto una imagen en sustitución del applet.

```
<APPLET CODE="NervousText.class" WIDTH=300 HEIGHT=50>  
<PARAM NAME="text" VALUE="JAVA es divertido">  
<IMG SRC="lastima.jpg" ALT="Se esta perdiendo muchas cosas porque  
su browser no soporta Java">  
</APPLET>
```

La evaluación del código **HTML** representado se realiza por etapas. En primer lugar se ha de representar el applet **NervousText** en un área de pantalla de 300 por 50 puntos de tamaño. Si esto no es posible, en su lugar se emitirá la imagen que se encuentra en el archivo **"lastima.jpg"**.

8. Aspectos de diseño

El código **HTML** que necesita para agregar applets a tus páginas es bastante simple. Si tienes experiencia con **HTML**, y has escrito tus propias páginas Web, no deberás tener problemas para agregar applets de Java a tu página. Pero también puede ser fácil sobrepasarse con los applets y olvidar que grandes cantidades de algo bueno pueden ocasionar problemas.

8.1. Estética de diseño

Los applets brindan nuevos aspectos estéticos a Web, como cualquier característica visual nueva. Al diseñar tus páginas tal vez quieras hacer una lista de las maneras en que los applets pueden mejorar tus páginas y de la manera en que pueden empeorarlas. Por ejemplo, los botones resplandecientes (que ya aprendimos a hacer en JavaScript) pueden agregar un bonito elemento visual a la página, pero algunas personas los pueden considerar excesivos.

8.2. Color

Recuerda que los applets agregan color a una página Web al igual que cualquier otro elemento gráfico. Si te preocupa mucho el color y

deseas integrar un esquema de colores en tus páginas web, trata de seleccionar applets que se ajusten a tu esquema de colores o que puedan ser personalizados para ajustarse a tu esquema de colores. Recuerda que un applet con un área negra grande puede confundirse con el fondo negro de una página web.

8.3. Apariencia

Algunos applets tienen apariencias distintas, tal vez en los gráficos y en ocasiones en la interfaz de usuario. Trata de seleccionar applets que mejoren la apariencia de tus páginas web sin ir en contra de ella. Por ejemplo, una marquesina con LEDs desplazantes puede ser excelente para anunciar nuevos grupos musicales en un club nocturno local, pero puede parecer poco apropiado para anunciar los servicios de una funeraria local. Los applets pueden crear diferentes impresiones a los usuarios y una mala primera impresión puede ser desastrosa para un sitio web.

8.4. Manía por la animación

La animaciones pueden ser una excelente manera de dar vida a una página web. También pueden ser una excelente manera de hacer que una página web sea más confusa. Aunque es posible emplear las animaciones de manera sutil para hacer que una página web resalte, es posible que se abuse de ellas como de cualquier otro efecto. Las técnicas de animación y Java pueden emplearse para crear cualquier cosa, desde botones que resalten al ser tocados hasta mascotas de página web completamente animadas, y ambos usos tienen su lugar en alguna parte de web. Sin embargo, una página con una barra de texto desplazante, botones móviles, un reloj y una mascota animada puede apabullar a los usuarios con demasiada información. Así como una página aburrida puede evitar que los usuarios regresen, una página atiborrada con demasiadas partes móviles puede asustar a los usuarios.

8.5. Diseño funcional

Además de cambiar el diseño visual de las páginas web, los applets también pueden cambiar el diseño funcional. Si tiene un applet en su página web que proporciona un servicio específico o ejecuta una tarea para el usuario, deberá tomar en cuenta algunos aspectos de diseño funcional.

8.6. Tamaño de los applets

Recuerde que debe proporcionar el tamaño inicial de un applet en el código HTML, que agrega un applet a su página. Este tamaño puede

ser una cantidad arbitraria, pero es probable que desee seleccionar un tamaño que maximice las funciones del applet en la mayoría de los casos. Supongamos que tiene un applet que pida al usuario que escriba información dentro de varios campos. Si sólo selecciona un tamaño arbitrario, los campos de entrada pueden quedar cortados. El resultado final es un applet menos efectivo. Lo mismo pasa con los applets diseñados para desplegar información, trátase de texto o imágenes. Debe dar a sus applets un tamaño que permita que información importante sea visible al abrir el applet, pero no debe ocupar todo el espacio en pantalla. Trate de lograr un balance entre el applet y el contenido de la página.

8.7. Cómo colocar applets

La colocación de los applets también se convierte en un aspecto importante al considerar la funcionalidad. Tal vez quiera colocar el applet de manera que se vea resaltado por el texto de la página o quizá desee poner el applet aparte del resto de la página usando otros elementos HTML o gráficos estándar. Sea cual sea la forma en que decida integrar sus applets a sus páginas web, no se extralimite añadiendo applets; vea el sitio como un todo y use los applets para mejorar sus páginas.

8.8. Sonido

Tal vez no parezca obvio que el sonido es un aspecto de diseño, pero hasta ahora el sonido no había sido un estorbo en World Wide Web. Hasta antes de Java, los sonidos de las páginas Web requerían que el usuario hiciera clic en un icono para activar un archivo de sonido, por lo que los usuarios nunca fueron sorprendidos por sonidos que salen de la nada. Java cambia esto. Con Java, puedes incorporar sonido en una página de manera que sea reproducido tan pronto como alguien accede a nuestra página. Debido a que mucha gente puede no darse cuenta de que esto es posible, también es posible que no tengan sus máquinas configuradas para manejar el sonido. Quizá sus niveles de volumen estén ajustados a un nivel muy alto o tal vez estén observando tu página desde un sitio donde no son bien vistos los sonidos extraños como la escuela, la Universidad o el trabajo.

El diseño de sonido no es tanto cosa de gusto como de buenos modales. Si el sonido mejora en gran medida nuestras páginas, no dejemos de usarlo, pero tratemos de hacerlo sabiamente. A continuación daremos algunos consejos para el empleo de sonidos en nuestras páginas:

- Evitemos los sonidos largos. Éstos pueden ocasionar que la gente se asuste mientras un sonido termina de reproducirse. La gente puede pasar apuros tratando de cubrir los altavoces de una máquina

mientras espera que un sonido largo acabe, Los sonidos largos también pueden incrementar el tiempo de transferencia y hacer que nuestras páginas sean lentas.

- Evitemos los sonidos rudos o muy fuertes. Así como un sonido largo puede ocasionar apuros, también puede ocasionarlos un sonido fuerte o rudo. Nadie quiere hacer clic en una página para escuchar un ruidoso martillo anunciando una página en construcción y seguro que nadie desea escuchar a su computadora insultando o haciendo sonidos inapropiados mientras observa una página rodeado de otras personas.

Así que si necesita usar el sonido, planéelo bien. Algunos sitios muy exitosos incorporan el sonido con buen gusto y de manera oportuna. Si no se usan con cuidado, los sonidos pueden perjudicar a un applet. Si emplea algo de cortesía y sentido común al diseñar sus páginas, éstas tendrán un atractivo aún mayor.

8.9. Evite atiborrarse de applets

Es muy fácil pasarse con los applets. Recuerda que no todas las páginas necesitan un applet y que muy pocas necesitan varios. Atiborrar una página con applets puede reducir el número de personas que regresen a nuestras páginas. El exceso de applets puede distraer y en ocasiones puede ser bastante molesto. Si has estado usando web durante mucho tiempo, tal vez recuerdes la etiqueta <BLINK> de Netscape. Lo que estaba diseñado simplemente para ser un medio para agregar énfasis, pronto se convirtió en una molestia en páginas web. Los applets de Java tienen el mismo potencial. Si se usan apropiadamente pueden lograr que su sitio sea excelente. Si exageramos su uso, podemos hacer que nuestro sitio sea de los menos populares.

8.10. Tiempo de transferencia

Colocar toneladas de applets en sus páginas web tal vez no se vea mal, pero puede reducir la velocidad de sus páginas hasta el punto de dejarlas inservibles. Puesto que los applets se bajan a través de Internet desde un servidor host, cada applet toma un poco de tiempo antes de poder empezar a ejecutarse. Para uno o dos applets, este tiempo realmente no es significativo (¡a menos que el applet sea gigantesco!) y es como cargar una imagen grande.

Sin embargo, si su página contiene varios applets, puede empezar a volverse lenta. Si los usuarios tienen que esperar 10 minutos para que se descargen todos los applets de su página, tal vez decidan que es mejor no esperar. Mucha gente paga por el tiempo de acceso y este tiempo es dinero. No hay muchas cosas en web por las que la gente esté

dispuesta a esperar tanto tiempo, así que trate de tomar en cuenta el tiempo de transferencia al diseñar sus páginas de applets.

E. ActiveX

1. Object Linking and Embedding (OLE) y klos controles ActiveX
 - 1.1. Los conceptos fundamentales de OLE
 - 1.2. Identificación universal única mediante GUIDs
 - 1.3. De OLE a ActiveX
 - 1.4. Ante todo, seguridad. Certificados firmados
2. La utilización de controles ActiveX
 - 2.1. La sintaxis HTML para controles ActiveX
 - 2.1.1. El tag <object>
 - 2.1.2. El tag <param>
 - 2.1.3. Otros elementos HTML
 - 2.2. Los controles ActiveX en el IE 3.0 y 4.0

D. ActiveX

1. Object Linking and Embedding (OLE) y los controles ActiveX

El desarrollo del lenguaje de programación Java perseguía el objetivo, tal y como hemos visto en el capítulo anterior, de crear programas ejecutables que se pueden transmitir a través de Internet. Durante el desarrollo de ActiveX se perseguía el mismo objetivo. Al contrario que en Sun, en Microsoft no se ha desarrollado ningún lenguaje de programación nuevo, sino que se han ampliado conceptos de modularización ya existentes.

1.1. Los conceptos fundamentales de OLE

En los lenguajes de programación de alto nivel, se distingue básicamente entre lenguajes intérprete y lenguajes de compilación. En el caso de los lenguajes intérprete, los comandos determinados por el programador se evalúan durante la ejecución del programa como activaciones de subprogramas. En cambio, los textos de programa escritos en lenguajes de compilación se traducen, antes de su ejecución, a programas en código máquina.

El texto de programa que resulta de esta traducción contiene de nuevo numerosas llamadas a rutinas de sistema que son, en definitiva, subprogramas del programa principal. En un segundo paso de trabajo, el llamado Linking, se enlazan tanto el programa traducido propiamente dicho como los subprogramas que allí se activan, para obtener un programa ejecutable. Esto tiene como consecuencia, que en caso de una actualización del lenguaje de programación, el programa tendrá que ser compilado de nuevo y también tendrá que ser distribuido otra vez entre los clientes de programa.

Esto es más problemático, tanto más como componentes de diferentes fabricantes de software hayan sido incorporados al producto definitivo, mediante el proceso del Linking. Microsoft inventó el concepto de las llamadas Dynamic Link Libraries para superar esta dificultad. Una Dynamic Link Library es un archivo de programa, que precisamente no se enlaza con el código ejecutable, sino que existe por separado en el disco duro del PC.

En general, se trata de archivos con la extensión de archivo .DLL. Respecto a la administración de memoria, este concepto de programación también es muy conveniente. El archivo DLL no se carga en la memoria principal (RAM) hasta que no se active una de las

funciones guardadas en él. También puede eliminarse de la memoria RAM, una vez haya cumplido su cometido. Pero lo más importante es que un archivo DLL puede modificarse por separado. Si, por ejemplo, se descubre un error en una rutina de sistema, es suficiente cambiar el archivo DLL en cuestión. En un caso como éste, los programas de aplicación que activan la rutina correspondiente no han de compilarse de nuevo y a pesar de ello, pueden sacar partido de las mejoras.

No obstante, para la realización consecuente del estilo de programación orientada a objetos, el concepto DLL es demasiado rígido. Más bien es deseable realizar el paradigma del objeto de programa también a nivel de sistema operativo. Las tecnologías desarrolladas por Microsoft para conseguir esa meta se llaman "Component Object Model" (COM) y "Object Linking and Embedding" (OLE).

Al principio OLE era tan sólo un protocolo estándar para la comunicación entre diferentes aplicaciones. Así, la versión original de OLE permitía, por ejemplo, incrustar tablas de Excel en textos creados con Word para Windows. Pero, con la especificación OLE2, este concepto se amplió considerablemente, de manera que también representa una base para objetos de programa más enrales. A pesar de ellos, OLE 2 tampoco representa un concepto orientado a objetos en sentido clásico. Las características esenciales de la orientación a objetos son la encapsulación de datos locales referidos a objetos, así como la herencia y el polimorfismo. Estos conceptos tienen un papel esencial en el diseño de lenguajes de programación de alto nivel como por ejemplo, C++. A nivel de sistema operativo, donde entran en juego estructuras de programa binarias, fueron necesarios puntos de partida diferentes.

Para evitar confusiones en este concepto, se introdujo un concepto nuevo e importante para los elementos del concepto OLE. Se trata del "Interface". En el marco del concepto OLE, el significado de una interfaz se corresponde aproximadamente con el de una función API en el marco del concepto DLL. Una interfaz es la unidad elemental más pequeña con la que se puede representar un objeto de programa OLE hacia el exterior. Una interfaz OLE se compone de varias, al menos tres, funciones de interfaz que a veces también se denominan métodos. Para encontrar cada una de estas funciones, cada objeto OLE precisa una tabla de asignación, que se llama VTABLE y contienen punteros a las direcciones de cada función.

1.2. Identificación universal única mediante GUIDs

GUID es la abreviatura de "Globally Unique Identifier" (identificador universal único). Cada objeto OLE tiene que poseer un GUID igual que cada una de sus interfaces. Existen, por tanto, Class-ID

e Interface-Ids (IID). La estructura de cada uno de estos identificadores es idéntica.

Un GUID se compone de un totalde 16 bytes o de 32 cifras hexadecimales. Normalmente, se dividen en cinco bloques. El primer bloque contiene 8 cifras, los bloques dos a cuatro, 4 cifras cada uno y el último bloque, las 12 cifras restantes del GUID. Los bloques se separan entre ellos mediante guiones. Un GUID típico podría tener el siguiente aspecto:

B16553C0-06DB-101B-85B2-0000C009BE81

Se tiene que poder asegurar que cualquier persona que desarrolle componentes OLE pueda crear prácticamente cualquier cantidad de GUIDs de estas características y utilizarlos en sus programas. Con este propósito, Microsoft ha desarrollado el programa GUIDGEN.EXE que se incluye como componente de todos los sistemas de desarrollo de software que vienen al caso.

Naturalmente, se plantea la cuestión de cómo podría funcionar todo esto. Lamentablemente, Microsoft no ha dado a conocer totalmente el modo de funcionar de este programa. La idea fundamental del programa, no obstante, se basa con toda seguridad en recopilar la mayor cantidad de informaciones individuales posibles, que se encuentran en el ordenador utilizado y que en su totalidad son forzosamente únicas en todo el mundo y utilizarlas como base para la generación del GUID. Las informaciones individuales que se plantean para este fin son, por ejemplo, el nombre del usuario, el número de identificación de la tarjeta de red que pueda existir, el número de bytes disponibles en el disco duro, la fecha y la hora.

1.3. De OLE a ActiveX

La especificación original de OLE conllevaba formaciones de código relativamente amplias para cada objeto. Por principio, con estos objetos de programa también era imaginable un intercambio a través de Internet. Los objetos ActiveX representan un avance respecto de la tecnología OLE, ya que crean módulos de código más reducidos. Como es sabido, la cuestión de la velocidad de transmisión de datos es el problema principal en internet. Por lo tanto y en general, es deseable que toda la información transmitida y por tanto, también los módulos de programa a transmitir, tengan el tamaño más pequeño posible.

¿Qué es lo que marca la diferencia entre los objetos OLE y los objetos ActiveX? La respuesta a esta pregunta es "realmente, nada de nada". Pero, sí que es necesario disponer de nuevas herramientas especiales apra el desarrollo de software, para poder cumplir la exigencia que acabamos de mencionar: obtener módulos binarios pequeños. En el desarrollo de un control OLE común, las herramientas

de desarrollo habituales, además de la interfaz indispensable IUnknown, implementan de manera estándar algunas otras interfaces como, por ejemplo, IDataObject e IoleControl. Esto, tras la especificación dperfeccionada "OLE Control Container Guidlines V 2.0", ya no es necesario. Los componentes ActiveX, de momento sólo tienen que disponer de la interfaz IUnknown.

Pero debido a la disminución de la cantidad de interfaces obligadas, la utilización de este tipo de objetos de programa se dificulta levemente. El rograma activador, que en el caso de objetos OLE correspondientes a las especificaciones anteriores, podía suponer la existencia de numerosas interfaces y métodos y ahora no dispone de ninguna información referente a las capacidades específicas de objeto.

Para superar esta dificultad se introdujo la llamda categoría de componentes. Utilizando las categorías de componentes, un objeto ActiveX puede representar su funcionalidad frente a un programa activador, tratándose de algo más que de una pura enumeración de las interfaces. De este modo, un programa puede informarse sobre las posibilidades del objeto en cuestión, sin tener que activar realmente el objeto.

1.4. Ante todo, seguridad. Certificados firmados

Una diferencia esencial entre el concepoto de los controles ActiveX y de los Java-Applets consiste en los conceptos de seguridad que se implementan en cada caso. Los Java-Applets, al igual que los programas Script, se ejecutan bajo total control del Web-Browser utilizado. Gracias a esto fue posible descargar completamente accesos al disco duro por parte de este tipo de programas y excluir el mayor riesgo de seguridad. No obstante, se tienen que indicar que al imposibilitar este tipo de funciones también se limita la capacidad de producción de los correspondientes conceptos del programa.

Los componentes ActiveX son de naturaleza binaria. Una vez un componente así queda instalado en un sistema y se inicia, por principio, su funcionamiento ya no es controlable desde el exterior. Para evitar daños en el sistema producidos por códigos ActiveX programados con mala fe, se precisa un concepto de seguridad completamente distinto al que se ha realizado en relación con Java. Esta nueva tecnología de seguridad se llama "Authenticode". En definitiva, se trata del principio de confianza. La base de la confianza a otorgar al programa en cuestión, la representan los llamados certificados firmados. Éstos se transmiten en forma de códigos de firmas digitales a prueba de falsificación al browser que ha exigido el control ActiveX.

En base a los elemento de diálogo contenidos en el certificado, el usuario puede decidir si asiente a la instalación del componente o no.

2. La utilización de controles ActiveX

Los controles ActiveX se incorporan a los documentos juntos con los elementos HTML, OBJECT y PARAM. A continuación, se comenta, en primer lugar la sintaxis general de esta vinculación y, después. La utilización pormenorizada de aquellos controles ActiveX que forman parte, por así decirlo, del equipamiento estándar de Internet Explorer.

2.1. La sintaxis HTML para controles ActiveX

El modo en que los controles ActiveX se incorporan a documentos HTML recuerda bastante a la utilización de los Java-Applets, de los que hemos hablado ya. Antes de describir detalladamente esta sintaxis, conviene remarcar de nuevo la diferencia básica entre ambas tecnologías.

Los Java-applets se transmiten en forma de códigos byte. Se trata de una compilación independiente de la plataforma, que tiene que ser interpretada por el browsers. Un Java-Applet no tiene valor sin el browsers.

En cambio, los controles ActiveX son objetos de programa compilados especialmente para sistemas de la familia Microsoft Windows. Después de la transmisión se instalan en el sistema, con lo que queda a disposición para usos posteriores. Los controles ActiveX también puede ser utilizados por otras aplicaciones Windows.

2.1.1. El tag <OBJECT>

El tag <OBJECT> pertenece a los últimos desarrollos en el ámbito HTML. Representa al margen de la introducción de los controladores de evento una parte fundamental del proyecto "Cougar", frente al HTML 3.2 actual, IE30 conoce este tag HTML que se presentan detalladamente a continuación.

ALIGN

El atributo Align sirve para determinar la posición relativa de la zona de la pantalla reservada para la representación del objeto, en relación a los elementos HTML que hay alrededor. Los posibles valores de atributo ya son conocidos.

CLASSID

El atributo CLASSID sirve para la identificación unívoca del objeto en cuestión. SI se trata de un control ActiveX, el correspondiente valor de atributo es una cadena de caracteres compuesta por la secuencia de caracteres "CLSID" seguida del llamado GUID. Esta abreviatura significa "Globally Unique Identifier". Para asegurar el carácter único y

universal de este tipo de números casi por principio aletatorio, se han previsto construcciones numéricas especialmente largas.

CODEBASE

El atributo CODEBASE contiene, por si fuera necesario, el URL de la carpeta, de la que se puede cargar el control ActiveX.

CODETYPE

El atributo CODETYPE especifica el "Internet Media Type" del archivo de código.

DATA

El valor del atributo opcional DATA también es un URL. Es este caso se trata de la dirección de un archivo de datos, al que, si fuera el caso, accede el objeto de programa.

DECLARE

El atributo DECLARE es un Flag. Ello significa que no se le tiene o no es necesario asignarle un valor. La palabra clave DECLARE tiene como efecto que, aunque el objeto correspondiente se declare, se evite que dicho objeto se ejecute inmediatamente.

STANDBY

Al atributo STANDBY se le puede asignar una cadena de caracteres como valor, que se muestra durante el proceso de carga, proceso que, en el caso de tener numerosos controles ActiveX, puede tardar bastante tiempo. La siguiente asignación de valor, por ejemplo sería adecuada:

STANDBY="Por favor, tenga paciencia"

TYPE

El atributo TYPE especifica el "Internet Media Type" del archivo cuyo URL fue determinado a través del atributo DATA.

HEIGHT, WIDTH, BORDER, HSPACE, VSPACE, USEPMAP, SHAPES, NAME, TITLE

Actúan de forma similar al resto de HTML

TABINDEX

El valor del atributo TABINDEX es un número entero, que indica la columna de tabulación en la que se ha de representar el objeto.

NOTAB

El atributo NOTAB es un flag, que determina que el objeto no está afectado por una disposición de tabulaciones quizá existente.

2.1.2. El tag <PARAM>

El tag HTML <PARAM> sólo está permitido en elementos APPLET o OBJECT. Anteriormente ya se ha descrito detalladamente en relación con los Java-applets. La sintaxis del tag <PARAM> si se utiliza dentro de un elemento OBJECT es idéntica a la sintaxis que allí se indicaba.

2.1.3. Otros elementos HTML

Dentro del elemento OBJECT pueden aparecer, al igual que en el caso de los elementos APLLET, tantos elementos HTML como se desee, que son considerados, sin embargo, como alternativa posiblemente necesaria al applet propiamente dicho. Esta funcionalidad, por cierto, se justifica en el hecho de que los browsers deben ignorar todos los tags, cuyo significado les sea desconocido.

Si un browser no reconoce el tag <OBJECT>, evaluará todos los elementos HTML que se encuentran entre las marcas <OBJECT> y </OBJECT> como si el elemento OBJECT no existiera. En cambio, los Browsers modernos, que pueden evaluar el elemento OBJECT, ignoran todos los tags HTML que se encuentran dentro de este elemento, a excepción del tag <PARAM>.

Apéndice A: HTML Language Reference

CONTENTS

- [Document Structure Elements](#)
- [Anchor Element](#)
- [Block Formatting Elements](#)
- [Character Data](#)
- [Document Sound](#)
- [Dynamic Documents](#)
- [Forms](#)
- [Inline Images](#)
- [Information-Type and Character-Formatting Elements](#)
- [List Elements](#)
- [Tables](#)

The vast range of HTML markup currently supported by available HTML user agents (Web browsers, such as Netscape, Mosaic, and so on) can be broadly divided into the following sections. Some elements described may not be supported by all browsers. Where an element is known to be supported by specific browsers, the element description will be labelled as such.

This appendix is divided into the following sections:

Document Structure Elements
Anchor Element
Block Formatting Elements
Character Data
Document Sound
Dynamic Documents
Forms
Inline Images
Information-Type and Character-Formatting Elements
List Elements
Tables

Note
<p>Stephen Le Hunte (cmlehunt@swan.ac.uk), the author of this appendix, is an independent software developer and freelance technical author specialising in HTML and WinHelp. He is currently studying for his Ph.D. at the University of Wales Swansea.</p>

Document Structure Elements

These elements are required within an HTML document. Apart from the prologue document identifier, they represent the only HTML elements that are explicitly required for a document to conform to the standard.

The essential document structure elements are

```
<HTML> . . . </HTML>
<HEAD> . . . </HEAD>
<BODY> . . . </BODY>
```

Prologue Identifiers

In order to identify a document as HTML, each HTML document should start with the prologue:

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">.
```

However, it is worth noting that if the document does not contain this type declaration, a browser should infer it. This document identifier identifies the document as conforming to the HTML 2.0 DTD.

```
<HTML> . . . </HTML>
```

The `<HTML>` element identifies the document as containing HTML elements. It should immediately follow the prologue document identifier, and it serves to surround all of the remaining text, including all other elements. Browsers use the presence of this element at the start of an HTML document to ensure that the document is actually HTML, according to the `text/html` MIME type. The document should be constructed like this:

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML>
  The rest of the document should be placed here.
</HTML>
```

The `HTML` element is not visible upon browser rendering and can contain only the `<HEAD>` and `<BODY>` elements.

```
<HEAD> . . . </HEAD>
```

The `<HEAD>` element of an HTML document is used to provide information about the document. It requires the `<TITLE>` element between `<HEAD>` and `</HEAD>` tags:

```
<HEAD>
  <TITLE>Introduction to HTML</TITLE>
</HEAD>
```

The `<HEAD>` and `</HEAD>` tags do not directly affect the look of the document when rendered.

The following elements are related to the `<HEAD>` element. Although they don't directly affect the look of the document when rendered, you can use them to provide important information to the browser. To do so, you employ the following elements, all of which should be included within the `<HTML> . . . </HTML>` tags.

<code><BASE></code>	Allows the base address of the HTML document to be specified
<code><ISINDEX></code>	Allows keyword searching of the document
<code><LINK></code>	Indicates relationships between documents
<code><META></code>	Specifies document information usable by server/clients
<code><NEXTID></code>	Creates unique document identifiers
<code><STYLE></code>	Specifies styles within the document when used by browsers that support use of style sheets
<code><TITLE></code>	Specifies the title of the document

Note

The <TITLE> element is the only element described here that is required as part of the <HEAD> of an HTML document for conformance to any HTML standard.

<BODY> . . . </BODY>

The <BODY> element of an HTML document, as its name suggests, contains all the text and images that make up the page, as well as all the HTML elements that provide the control and formatting of the page. The format is

```
<BODY>
  The rest of the document included here
</BODY>
The <BODY>...</BODY> tags should be directly enclosed
within the <HTML>...</HTML> tags.
```

The <BODY> and </BODY> tags themselves do not directly affect the look of the document when rendered, but they are required in order for the document to conform to the specification standard. Various attributes of the opening <BODY> tag can be used to set up various page-formatting settings.

The capability to specify background images and colors for HTML documents was first implemented by Netscape and has since been implemented by most other browsers. It should be noted that the following elements might not be supported by every browser.

BACKGROUND

Recent versions of the proposed HTML 3.2 specification have added a BACKGROUND attribute to the <BODY> element. The purpose of this attribute is to specify a URL pointing to an image that is to be used as a background for the document. In most browsers, this background image is used to tile the full background of the document-viewing area. Consider the following code:

```
<BODY BACKGROUND="imagenname.gif">
  Rest of the document goes here
</BODY>
```

It would cause whatever text, images, and so on that appeared in the body of the document to be placed on a background consisting of the imagenname.gif graphics file, being tiled to cover the viewing area (like bitmaps are used for Windows wallpaper). Most browsers that support this attribute allow the use of GIF and JPG images for document backgrounds, whereas Internet Explorer supports those, plus Windows BMP files.

BGCOLOR

The BGCOLOR attribute for <BODY> is not currently in the proposed HTML 3.2 specification, but is supported by Netscape, the Internet Explorer, NCSA Mosaic, and many other browsers and is being considered for inclusion in HTML 3.2. It allows the setting of the color of the background without having to specify a separate image that requires another network access to load. The format is

```
<BODY BGCOLOR="#rrggbb">
  Rest of document goes here
</BODY>
```

where #rrggbb is a hexadecimal (base 16) red-green-blue triplet used to specify the background color.

Recently, browsers have begun allowing the use of special names to define certain colors. Appendix E, "Colors by Names," presents a list of all the color names recognized by popular browsers and also includes their corresponding hexadecimal triplet values.

Note that using color names is browser specific, so you have greater control over the displayed colors if you use the #rrggbb values instead.

If you change the background colors or patterns within a presentation, remember to verify that the foreground still looks good on the new background.

Color Considerations
Most graphical browsers allow the downloading of embedded images to be turned off to allow for faster downloading and display of the HTML document. If you turn off downloading for embedded images, background images will not be loaded or displayed. If this happens and no BGCOLOR attribute was specified, all of the foreground text and link-color attributes (TEXT, LINK, VLINK, and ALINK) will be ignored. This is so that documents are not rendered illegibly if the text color scheme authored for use over the set image clashes with the default browser background.

BGPROPERTIES

In Internet Explorer, you can watermark HTML documents by fixing a background image so that it doesn't scroll as a normal background image does. To give a page with a background image a watermarked background, add BGPROPERTIES=FIXED to the <BODY> element as this code shows:

```
<BODY BACKGROUND="filename.gif" BGPROPERTIES=FIXED>
```

LEFTMARGIN

This Internet Explorer attribute allows you to set the left margin of the document:

```
<BODY LEFTMARGIN="40">This document is indented 40 pixels  
from the left hand  
edge of the browser window</BODY>
```

If you set LEFTMARGIN to 0, the page will start at the left-hand side of the page.

LINK, VLINK, and ALINK

These link attributes allow you to control the color of link text. VLINK stands for *visited link*, and ALINK stands for *active link* (this sets the color that the link text will be for the time that it is clicked on). Generally, the default colors of these attributes are LINK=blue (#0000FF), VLINK=purple (#800080), and ALINK=red (#FF0000). The format for these attributes is the same as that for BGCOLOR and TEXT:

```
<BODY LINK="#rrggbb" VLINK="#rrggbb" ALINK="#rrggbb">  
  Rest of document goes here  
</BODY>
```

You also can use color names rather than hexadecimal values for these attributes. See Appendix E for a complete list of color names and their hexadecimal values.

TEXT

The TEXT attribute can be used to control the color of all the normal text in the document. This basically consists of all text that is not specially colored to indicate a link. The format of TEXT is the same as that of BGCOLOR:

```
<BODY TEXT="#rrggbb">  
  Rest of document goes here  
</BODY>
```

You also can use color names rather than hexadecimal values for these attributes. See Appendix

E for a complete list of color names and their hexadecimal values.

TOPMARGIN

This Internet Explorer-specific attribute allows the top margin of the document to be set:

```
<BODY TOPMARGIN="40">This document is indented 40 pixels  
from the top hand  
edge of the browser window</BODY>
```

If you set TOPMARGIN to 0, the page will start at the very top of the page.

<BASE . . .>

The <BASE . . .> element allows you to set the URL of the document itself, to help browsers in situations where the document might be read out of context. It is especially useful in allowing browsers to determine any partial URLs or relative paths that might be specified (for example, in <A HREF> elements or in paths used to specify (images)). The <BASE> element should appear within the bounds of the <HEAD> element only.

Where the base address is not specified, the browser uses the URL it used to access the document to resolve any relative URLs.

HREF

The <BASE> element has one standard attribute, HREF, that identifies the URL. The URL should be fully qualified, as in this example:

```
<BASE HREF="http://www.myhost.com/">
```

This code specifies www.myhost.com to be the base from which all relative URLs should be determined.

TARGET

Netscape 2.0 and Internet Explorer 3.0 add one other attribute to the <BASE> element. With the introduction of targeted windows, you can use the TARGET attribute as you use it in anchors (<A>). This allows you to pick a default-named target window for every link in a document that does not have an explicit TARGET attribute. Its format is

```
<BASE TARGET="default_target">
```

<ISINDEX . . .>

The <ISINDEX> element tells the browser that the document is an index document. As well as reading it, the reader can use a keyword search.

Readers can query the document with a keyword search by adding a question mark to the end of the document address, followed by a list of keywords separated by plus signs.

Note

The <ISINDEX> element usually is generated automatically by a server. If added manually to an HTML document, the browser assumes that the server can handle a search on the document. To use the <ISINDEX> element, the server must have a search engine that supports this element.

ACTION

Netscape provides the ACTION attribute for the <ISINDEX> element. When used in the <ISINDEX> element, it explicitly specifies the CGI script or program to which the text string in the input box should be passed. For example,

```
<ISINDEX ACTION="Websearch">
```

passes the text entered into the input box on the page to the CGI script Websearch.

Note
Websearch in the preceding example is a hypothetical CGI script. The ACTION attribute must point to a properly configured script on the host machine.

PROMPT

Netscape provides the PROMPT attribute for the <ISINDEX> element. PROMPT allows you to specify text that should be placed before the text-input field of the index. The syntax is

```
<ISINDEX PROMPT="Any_text_string: ">
```

where Any_text_string is the text you want to be displayed before the input box.

<LINK...>

The <LINK> element indicates a relationship between the document and some other object. A document may have any number of <LINK> elements.

The <LINK> element is empty (it does not have a closing element) but takes the same attributes as the Anchor element (for example, REL, REV, METHODS, TITLE, HREF, and so on).

The <LINK> element typically would be used to provide pointers to related indexes or glossaries. Links also can be used to indicate a static tree structure in which the document was authored by pointing to a parent, next, and previous document, for example.

Servers also may allow links to be added by those who do not have the right to alter the body of a document.

The <LINK> element represents one of the primary style sheet inclusion mechanism elements. It can be used to specify the location of the style sheet that is to be used for the document. For example:

```
<HTML>
<HEAD>
<TITLE>This HTML document uses a style sheet</TITLE>
<LINK REL="stylesheet" TYPE="text/css"
HREF="http://www.stylesheets.com/sheets/formal.css"
TITLE="formal">
</HEAD>
<BODY>
  Rest of the document goes here
</BODY>
</HTML>
```

In the preceding HTML fragment, the <LINK> element points to the file formal.css at the given URL. It tells the browser that

- The file addressed is a style sheet, by explicitly giving the text/css MIME type.
- The file's relationship to the HTML document is that it is a stylesheet.
- The stylesheet's TITLE is formal.

Note
This HTML fragment represents part of a <i>work in progress</i> specification

of the World Wide Web Consortium (W3C).

For more information about these specific attributes, see the <A> section; for more general information about style sheets, see the style sheets section.

<NEXTID . . . >

The <NEXTID> element, included in old HTML specifications, is not widely supported and its use is not recommended. Previously, it could be used to provide information about the name of new <A> elements when a document was being edited.

<TITLE> . . . </TITLE>

Every HTML document must have a <TITLE> element. As its name suggests, it is used to specify the title of the document in question. Unlike headings, titles typically are not rendered in the text of a document itself. Normally, browsers will render the text contained within the <TITLE> . . . </TITLE> elements in the title bar of the browser window.

The <TITLE> element must occur within the head of the document and may not contain anchors, paragraph elements, or highlighting. Only one title is allowed in a document.

Note

Although the length of the text specified in the <TITLE> . . . </TITLE> elements is unlimited, for display reasons, most browsers will truncate it. For this reason, title text should be kept short but should be enough to uniquely identify the document. A short title such as *Introduction* may be meaningless out of context, for example, but if the title were *An Introduction to HTML elements*, it would be obvious what the document is about.

This is the only element that is required within the <HEAD> element.

```
<HEAD>
  <TITLE>Welcome to the HTML Reference</TITLE>
</HEAD>
```

<META . . . >

The <META> element is used within the <HEAD> element to embed document metainformation not defined by other HTML elements. Such information can be extracted by servers/clients for use in identifying, indexing, and cataloging specialized document metainformation.

Although it generally is preferable to use named elements that have well-defined semantics for each type of metainformation, such as title, this element is provided for situations where strict SGML parsing is necessary and the local DTD is not extensible.

In addition, HTTP servers can read the content of the document head to generate response headers corresponding to any elements defining a value for the attribute HTTP-EQUIV. This gives document authors a mechanism (not necessarily the preferred one) for identifying information that should be included in the response headers for an HTTP request.

Attributes of the <META> element are listed in the following sections.

CONTENT

The metainformation content to be associated with the given name and/or HTTP response header.

If the document contains

```
<META HTTP-EQUIV="Expires" CONTENT="Sat, 06 Jan 1990
```

```
00:00:01 GMT">
<META HTTP-EQUIV="From" CONTENT="nick@htmlib.com">
<META HTTP-EQUIV="Reply-to" CONTENT="stephen@htmlib.com">
```

the HTTP response header would be

```
Expires: Sat, 06 Jan 1990 00:00:01 GMT
From: nick@htmlib.com
Reply-to: stephen@htmlib.com
```

Commonly, HTML documents can be seen to contain a listing of repeated terms. Some Web search/indexing engines use the keywords information generated from the server or from those specified in the `<META HTTP-EQUIV="Keywords" CONTENT="...">` markup to determine the content of the specified document and to calculate their *relevance rating* (how relevant the document is to the specific search string) for the search results.

When the `HTTP-EQUIV` attribute is not present, the server should not generate an HTTP response header for this metainformation. For example,

```
<META NAME="IndexType" CONTENT="Service">
```

Do *not* use the `<META>` element to define information that should be associated with an existing HTML element.

The following is an inappropriate use of the `<META>` element:

```
<META NAME="Title" CONTENT="Welcome to the HTML
Reference">
```

Do *not* name an `HTTP-EQUIV` equal to a responsive header that typically should be generated only by the HTTP server. Some inappropriate names are `Server`, `Date`, and `Last-modified`. Whether a name is inappropriate depends on the particular server implementation. It is recommended that servers ignore any `<META>` elements that specify HTTP-equivalents (that are not case sensitive) equal to their own reserved response headers.

The `<META>` element is particularly useful for constructing dynamic documents via the client pull mechanism. This uses the following syntax:

```
<META HTTP-EQUIV="Refresh" CONTENT="x">
```

This causes the browser to believe that the HTTP response when the document was retrieved from the server included the following header:

```
Refresh: x
```

It also causes the document to be reloaded in *x* seconds.

Note
In the preceding example, when the document refreshes, loading itself, the browser will infinitely reload the same document over and over. The only way out of this situation is for the user to activate some hyperlink on the page, load a different document, or click the Back button to reload a previous document.

This can be useful to provide automatic redirection of browsers. If the element is

```
<META HTTP-EQUIV="Refresh" CONTENT="2;
URL=http://some.site.com/otherfile.html">
```

the `Refresh` directive would cause the file at `http://some.site.com/otherfile.html` to be loaded after two seconds. Although

this generally works if the URL specified is partial, you should use a fully qualified URL to ensure its proper functioning.

HTTP-EQUIV

This attribute binds the element to an HTTP response header. If the semantics of the HTTP response header named by this attribute is known, then the contents can be processed based on a well-defined syntactic mapping whether or not the DTD includes anything about it. HTTP header names are not case sensitive. If not present, the `NAME` attribute should be used to identify this meta-information, and it should not be used within an HTTP response header.

NAME

Metainformation name. If the `NAME` attribute is not present, the name can be assumed to be equal to the value `HTTP-EQUIV`.

Anchor Element

The anchor text is probably the single most useful HTML element. It is the element that is used to denote hyperlinks-the entire essence of HTML as a hypertext application.

`<A . . . > . . . `

Anchor elements are defined by the `<A>` element. The `<A>` element accepts several attributes, but either the `NAME` or `HREF` attribute is required.

Attributes of the `<A>` element are described in the following sections.

HREF

If the `HREF` (hypertext reference) attribute is present, the text between the opening and closing anchor elements becomes a hypertext link. If this hypertext is selected by readers, they are moved to another document or to a different location in the current document whose network address is defined by the value of the `HREF` attribute. Typically, hyperlinks specified using this element would be rendered in underlined blue text, unless the `LINK` attribute of the `<BODY>` element has been specified.

In this example, selecting the text `HTMLib` takes the reader to a document located at <http://www.htmlib.com>:

```
See <A HREF="http://www.htmlib.com/">HTMLib</A> for more
information about the
HTML Reference.
```

With the `HREF` attribute, the form `HREF="#identifier"` can refer to another anchor in the same document or to a fragment of another document that has been specified using the `NAME` attribute.

In this example, `<` and `>` are character data elements that render as `<` and `>`, respectively. In this case, they are used so that `<PRE>` is actually rendered on-screen (so that the browser doesn't think that the following text is preformatted text).

```
The <A HREF="document.html#pre">&lt;PRE&gt;</A> provides
details about the
preformatted text element.
```

Selecting the link takes the reader to another anchor (that is, `<PRE>`) in a different document (`document.html`). If the anchor is in another document, the `HREF` attribute may be relative to the document's address or the specified base address, or it can be a fully qualified URL.

Table B.1. Several other forms of the HREF attribute permitted by browsers.

	Makes a link to another document located on a World Wide Web server.
	Makes a link to an FTP site. Within an HTML document, normally a connection to an anonymous FTP site would be made. Some browsers, however, allow connections to private FTP sites. In this case, the anchor should take the form ftp://lehunte@htmlib.com and the browser then prompts the user for a password for entry to the site.
	Makes a link to a Gopher server.
	Activating such a link brings up the browser's mailing dialog box (if it has mailing capabilities; otherwise, whatever default e-mail software is installed on the system should be activated), allowing the user to send mail messages to the author of the document, or whoever's address is specified in the <code>mailto:</code> attribute. NCSA Mosaic supports use of the <code>TITLE</code> attribute for the anchor element when used with <code>mailto:</code> links. It allows the author to specify the subject of the mail message that will be sent. Netscape allows specification of the subject line by using the following syntax: link text
	Makes a link to a Usenet newsgroup. Care should be taken in using such links because the author cannot know what newsgroups are carried by the local news server of the user.
	Makes a link to a specific newsrsrc file. The newsrsrc file is used by Usenet news reading software to determine what groups carried by the news server the reader subscribes to.
	Specifies a different news server to that which the user may normally use.
	Activating such a link initiates a Telnet session (using an external application) to the machine specified after the <code>telnet://</code> label.
	Makes a link that connects to a specified WAIS index server.

METHODS

The `METHODS` attributes of anchors and links provide information about the functions the user may perform on an object. These are more accurately given by the HTTP protocol when it is used, but it may be useful to include the information in advance in the link. For example, the browser may chose a different rendering as a function of the methods allowed-something that is searchable may get a different icon or link text display method.

The value of the `METHODS` attribute is a comma-separated list of HTTP methods supported by the object for public use.

NAME

If present, the `NAME` attribute allows the anchor to be the target of a link. The value of the `NAME` attribute is an identifier for the anchor, which may be any arbitrary string but must be unique

within the HTML document.

```
<A NAME="pre">&lt;PRE&gt;</A> gives information about...
```

Another document then can make a reference explicitly to this anchor by putting the identifier after the address, separated by a hash sign:

```
<A HREF="document.html#pre">
```

REL

The REL attribute gives the relationship(s) described by the hypertext link from the anchor to the target. The value is a comma-separated list of relationship values, which will have been registered by the HTML registration authority. The REL attribute is used only when the HREF attribute is present.

REV

The REV attribute is the same as the REL attribute, but the semantics of the link type are in the reverse direction. A link from A to B with REL="X" expresses the same relationship as a link from B to A with REV="X". An anchor may have both REL and REV attributes.

TARGET

With the advent of frame page formatting, browser windows now can have names associated with them. Links in any window can refer to another window by name. When you click on the link, the document you asked for appears in that named window. If the window is not already open, Netscape opens and names a new window for you.

The syntax for the targeted windows is

```
<A HREF="download.html" TARGET="reference">Download  
information</A>
```

This loads the document `download.html` in the frame that has been designated as having the name `reference`. If no frame has this name, Netscape opens a new browser window to display the document.

Note

The use of targeted browser windows is supported by those browsers that currently support the use of <FRAME> page layout (Netscape and Internet Explorer). If the targetted document is part of a frameset, various reserved names can be used to allow smooth window transition. For more information, see <FRAMES>.

TITLE

The TITLE attribute is informational only. If present, the TITLE attribute should provide the title of the document whose address is given by the HREF attribute.

This might be useful because it allows the browser to display the title of the document being loaded as retrieval starts-providing information before the new document can be viewed. It is up to individual browsers to specify how they display the title information, but usually it is displayed in the title bar of the browser window. Some documents (such as Gopher or FTP directory listings) do not themselves contain title information within the document. The TITLE attribute can be used to provide a title to such documents. As mentioned earlier, Mosaic supports use of the TITLE attribute to specify the subject of a mail message sent when the user activates a link.

URN

If present, the URN attribute specifies a uniform resource name (URN) for a target document. The precise specification for URN has not yet been defined, so its use is not recommended.

Block Formatting Elements

Block formatting elements are used for the formatting of whole blocks of text within an HTML document (instead of single characters). They should all (if present) be within the body of the document (that is, within the `<BODY> . . . </BODY>` elements).

The essential block formatting elements follow:

<code><ADDRESS> . . . </ADDRESS></code>	Formats an address section
<code><BASEFONT SIZE=...></code>	Specifies the default font size for the document
<code><BLOCKQUOTE> . . . </BLOCKQUOTE></code>	Quotes text from another source
<code>
</code>	Forces a line break
<code><CENTER> . . . </CENTER></code>	Centers text on the page
<code><COMMENT> . . . </COMMENT></code>	Encloses text as a comment
<code><DFN> . . . </DFN></code>	Defines an instance
<code><DIV> . . . </DIV></code>	Allows centering or left/right justification of text
<code> . . . </code>	Sets/changes the font size, color, and type
<code><HR></code>	Renders a sizeable hard line on the page
<code><Hx> . . . </Hx></code>	Formats six levels of heading
<code><LISTING> . . . </LISTING></code>	Formats text
<code><MARQUEE></code>	Highlights scrolling text
<code><NOBR></code>	Specifies that words aren't to be broken
<code><P> . . . </P></code>	Specifies what text constitutes a paragraph and its alignment
<code><PLAINTEXT></code>	Formats text
<code><PRE> . . . </PRE></code>	Uses text already formatted
<code><WBR></code>	Specifies that a word is to be broken if necessary
<code><XMP> . . . </XMP></code>	Formats text

`<ADDRESS> . . . </ADDRESS>`

As its name suggests, the `<ADDRESS> . . . </ADDRESS>` element can be used to denote information such as addresses, authorship credits, and so on.

Typically, an address is rendered in an italic typeface and may be indented, although the actual implementation is at the discretion of the browser. The `<ADDRESS>` element implies a paragraph break before and after, as shown in this code:

```
<ADDRESS>
Mr. Cosmic Kumquat<BR>
SSL Trusters Inc.<BR>
1234 Squeamish Ossifrage Road<BR>
Anywhere<BR>
NY 12345<BR>
U.S.A.
</ADDRESS>
```

`<BASEFONT ...>`

Changes the size of the `<BASEFONT>`, on which all relative `` changes are based. It defaults to 3 and has a valid range of 1-7.

```
<BASEFONT SIZE=5>
```

FACE

Changes the face of the HTML document <BASEFONT>, exactly as it works for .

Note

This attribute is Internet Explorer specific.

COLOR

Allows the <BASEFONT> color for the HTML document to be set. Colors can be set by using one of the reserved color names or as an rrggbb hexadecimal triplet value.

Note

The <BASEFONT SIZE=...> element is supported only by Netscape and the Internet Explorer, with the ...FACE and ...COLOR attributes being Internet Explorer specific. This kind of presentation markup also can be specified within a style sheet.

<BLOCKQUOTE>...</BLOCKQUOTE>

The <BLOCKQUOTE> element can be used to contain text quoted from another source.

Typically, <BLOCKQUOTE> rendering would be a slight extra left and right indent, and possibly rendered in an italic font. The <BLOCKQUOTE> element causes a paragraph break and provides space above and below the quotation.

```
In "Hard Drive", a former Microsoft project manager has
said,
<BLOCKQUOTE>
"Imagine an extremely smart, billionaire genius who is 14
years old and subject to temper tantrums"
</BLOCKQUOTE>
```


The line break element specifies that a new line must be started at the given point. The amount of line space used is dependent on the particular browser, but is generally the same as it would use when wrapping a paragraph of text over multiple lines.

Note

Some browsers may collapse repeated
 elements and render as if only one had been inserted, as shown in this example:

```
<P>
Mary had a little lamb<BR>
Its fleece was white as snow<BR>
Everywhere that Mary went<BR>
She was followed by a little lamb.
```

With the addition of floating images (an embedded image aligned to the left or right of the browser display window, with text flowing around the image), it became necessary to expand the
 element. Normal
 still just inserts a line break. A CLEAR attribute was added to
, so

CLEAR=left will break the line and move vertically down until you have a clear left margin (where there are no floating images).

CLEAR=right does the same for the right margin.

CLEAR=all moves down until both margins are clear of images.

The `CLEAR` attribute (as well as floating images) currently are supported only by Netscape and the Internet Explorer.

<CENTER>

All lines of text between the begin and end of the `<CENTER>` element are centered between the current left and right margins. This element was introduced by the Netscape authors because it was claimed that using `<P ALIGN=--CENTER-->` "broke" existing browsers when the `<P>` element was used as a container (that is, with a closing `</P>` element).

The element is used as shown here, and any block of text (including any other HTML elements) can be enclosed between the centering elements:

```
<CENTER>All this text would be centered in the
page</CENTER>
```

Note

Most browsers will internally work around this element to produce the desired format, but it is an element introduced by Netscape authors.

<COMMENT> . . . </COMMENT>

The `<COMMENT>` element can be used to comment out text. As such, it is similar to the `<!-- . . . -->` element.

Any text placed between the `<COMMENT>` and `</COMMENT>` elements will not render on-screen, allowing comments to be placed in HTML documents. For example,

```
<COMMENT>This text won't render. I can say what I like
here, it won't appear
</COMMENT>
```

would not render on-screen.

Note

This element is supported only by Internet Explorer and Mosaic.

<DFN> . . . </DFN>

Use of the `<DFN>` element currently is supported only by Internet Explorer.

The `<DFN>` element can be used to mark the defining instance of a term—for example, the first time some text is mentioned in a paragraph.

Typically, it will render italicized.

```
The <DFN>Internet Explorer</DFN> is Microsoft's Web
browser.
```

for example, renders as

The *Internet Explorer* is Microsoft's Web browser.

<DIV> . . . </DIV>

Note

Use of the `<DIV>` element currently is supported only by Netscape (after version 2.0).

The `<DIV>` element, as described in the HTML 3.2 specification, should be used with a `CLASS` attribute to name a section of text as being of a certain style as specified in a style sheet.

Netscape has implemented the <DIV> element to work as the <P ALIGN= ...> element. Essentially, text surrounded by the <DIV>...</DIV> elements is formatted according to the description attached to the ALIGN attribute within the <DIV> elements, as shown in this example:

```
<DIV ALIGN="left">This text will be displayed left aligned  
in the browser  
window.</DIV>  
<DIV ALIGN="center">This text will be centered.</DIV>  
<DIV ALIGN="right">This text will be displayed aligned to  
the right of the  
browser window.</DIV>
```


Netscape 1.0 (and above) and Microsoft's Internet Explorer support different-sized fonts within HTML documents. This should be distinguished from headings.

The element is . Valid values range from 1-7. The default FONT size is 3. The value given to SIZE optionally can have a + or - character in front of it to specify that it is relative to the document <BASEFONT>. The default <BASEFONT SIZE= ...> is 3 and is specified with the <BASEFONT SIZE ...> element.

```
<FONT SIZE=4>changes the font size to 4</FONT>
```

```
<FONT SIZE=+2>changes the font size to BASEFONT SIZE ... +  
2</FONT>
```

Note

The element currently is supported only by Netscape and Internet Explorer.

Microsoft's Internet Explorer supports the capability to change the font color as well as the typeface. It adds COLOR and FACE attributes to the element. Netscape supports the use of the COLOR attribute only.

COLOR = #rrggbb OR COLOR = color

The COLOR attribute sets the color that text appears on-screen. #rrggbb is a hexadecimal color denoting an RGB color value. Alternatively, the color can be set to one of the available predefined colors. These color names can be used for the BGCOLOR, TEXT, LINK, ALINK, and VLINK attributes of the <BODY> tag as well.

```
<FONT COLOR="#ff0000">This text is red.</FONT>
```

or

```
<FONT COLOR="Red">This text is also red.</FONT>
```

Note

The use of names for coloring text currently is supported only by the Microsoft Internet Explorer and Netscape. Also, it should be noted that HTML attributes of this kind (that format the presentation of the content) also can be controlled via the use of style sheets.

FACE=name [, name] [, name]

The FACE attribute sets the typeface used to display the text on-screen. The typeface displayed already must be installed on the user's computer. Substitute typefaces can be specified in case the chosen typeface is not installed on the user's computer. If no exact font match can be found, the

text is displayed in the default type that the browser uses for displaying normal text.

```
<, Comic Sans MS"> This text will be displayed in either  
Courier New, or Comic Sans MS, depending on which fonts  
are installed on the browser's system. It will use the  
default 'normal' font if neither is installed.  
</FONT>
```

Note

When using this element, care should be taken to try to use font types that will be installed on the user's computer if you want the text to appear as desired. Changing the font is Internet Explorer specific and also can be set within a style sheet.

<HR>

A horizontal rule (<HR>) element is a divider between sections of text, such as a full-width horizontal rule or a similar graphic.

```
<HR>  
<ADDRESS>April 12, 1996, Swansea</ADDRESS>  
</BODY>
```

The <HR> element specifies that a horizontal rule of some sort (the default is a shaded, engraved line) be drawn across the page. It is possible to control the format of the horizontal rule.

<HR ALIGN=left | right | center>

Because horizontal rules do not have to be the width of the page, it is necessary to allow the alignment of the rule to be specified. Using these values, rules can be set to display centered, left, or right aligned.

<HR COLOR=name | #rrggbb>

Internet Explorer enables you to specify the hard rule color. Accepted values are any of the Internet Explorer supported color names or any acceptable rr gg bb hexadecimal triplet.

<HR NOSHADE>

For those times when a solid bar is required, the NOSHADE attribute lets you specify that the horizontal rule should not be shaded at all.

<HR SIZE=number>

The SIZE attribute lets you specify the thickness of the horizontal rule. The *number* value specifies how thick the rule will be in pixels.

<HR WIDTH=number | percent>

The default horizontal rule is always as wide as the page. With the WIDTH attribute, you can specify an exact width in pixels or a relative width measured in percent of the browser display window.

<Hx> . . . </Hx>

HTML defines six levels of heading. A heading element implies all the font changes, paragraph breaks before and after, and white space necessary to render the heading.

The highest level of headings is <H1>, followed by <H2> . . . <H6>.

An example follows:

```
<H1>This is a first level heading heading</H1>
Here is some normal paragraph text
<H2>This is a second level heading</H2>
Here is some more normal paragraph text.
```

The rendering of headings is determined by the browser, but typical renderings (as defined in the HTML 2.0 specification) follow:

<code><H1>...</H1></code>	Bold, very large font, centered. One or two blank lines above and below.
<code><H2>...</H2></code>	Bold, large font, flush left. One or two blank lines above and below.
<code><H3>...</H3></code>	Italic, large font, slightly indented from the left margin. One or two blank lines above and below.
<code><H4>...</H4></code>	Bold, normal font, indented more than H3. One blank line above and below.
<code><H5>...</H5></code>	Italic, normal font, indented as H4. One blank line above.
<code><H6>...</H6></code>	Bold, indented same as normal text, more than H5. One blank line above.

Note

These heading alignments can be overridden by using `<CENTER>` elements or by `ALIGN`ing the heading.

Although heading levels can be skipped (for example, from H1 to H3), this practice is not recommended because skipping heading levels may produce unpredictable results when generating other representations from HTML. For example, much talked about automatic contents/index generation scripts could use heading settings to generate contents trees where `<H2>` would be considered to label the start of a section that is a subsection of a section denoted by an `<H1>` element, and so on.

Included in the HTML 3.2 specification is the capability to align headings.

`ALIGN=left|center|right` can be added to the `<H1>` through to `<H6>` elements. For example,

```
<H1 ALIGN=center>This is a centered heading</H1>
```

aligns a heading of style 1 in the center of the page.

Note

This element currently is supported only by Mosaic and Netscape. The Internet Explorer supports only the `center` value, centering the heading.

<LISTING>...</LISTING>

The `<LISTING>` element can be used to present blocks of text in fixed-width font, so it is suitable for text that has been formatted on-screen. As such, it is similar to the `<PRE>` and `<XMP>` element but has a different syntax.

Typically, it renders as fixed-width font with white space separating it from other text. It should be rendered so that 132 characters fit on the line.

Note

Only Netscape actually complies with this.

The code

```
Some might say<LISTING>that two heads</LISTING>are better
than one
```

renders as

Some might say
that two heads
are better than one.

Note

The Internet Explorer and Netscape translate any special characters included within `<LISTING>` elements. If characters such as `<`, `>`, and so on are used, they are translated to `<` and `>`. Mosaic treats the text contained within the elements literally.

`<MARQUEE> . . . </MARQUEE>`

Note

This element currently is supported only by Microsoft Internet Explorer.

The `<MARQUEE>` element allows you to create a region of text that can be made to scroll across the screen (much like the Windows Marquee screen saver):

```
<MARQUEE>This text will scroll from left to right  
slowly</MARQUEE>
```

ALIGN

This attribute can be set to `TOP`, `MIDDLE`, or `BOTTOM` and specifies that the text around the marquee should align with the top, middle, or bottom of the marquee.

```
<MARQUEE ALIGN=TOP>Hello in browser land.</MARQUEE>Welcome  
to this page
```

The text `Welcome to this page` is aligned with the top of the marquee (which scrolls the text `Hello in browser land` across the screen).

Note

Until the marquee width is limited by setting the `WIDTH` attribute, the marquee occupies the whole width of the browser window, and any following text is rendered below the marquee.

BEHAVIOR

This can be set to `SCROLL`, `SLIDE`, or `ALTERNATE`. It specifies how the text displayed in the marquee should behave. `SCROLL` (the default) makes the marquee text start completely off one side of the browser window, scroll all the way across and completely off the opposite side, and then start again. `SLIDE` causes the text to scroll in from one side of the browser window and then stick at the end of its scroll cycle. `ALTERNATE` means bounce back and forth within the window.

```
<MARQUEE BEHAVIOR=ALTERNATE>This marquee will "bounce"  
across the screen</  
MARQUEE>
```

BGCOLOR

This specifies a background color for the marquee, as an `rrggb` hexadecimal triplet or as one of the reserved color names. (See `<BODY BGCOLOR>` for more information.)

DIRECTION

This specifies in which direction the `<MARQUEE>` text should scroll. The default is `LEFT`, which

means that the text will scroll to the left from the right-hand side of the marquee. This attribute also can be set to `RIGHT`, which causes the marquee text to scroll from the left to the right.

HEIGHT

This specifies the height of the marquee, in pixels (`HEIGHT=n`) or as a percentage of the screen height (`HEIGHT=n%`).

HSPACE

This attribute is the same as that for `` (images). It specifies the number of pixels of free space at the left- and right-hand sides of the marquee so that the text that flows around it doesn't push up against the sides.

LOOP

`LOOP=n` specifies how many times a marquee loops when activated. If `n=-1` or `LOOP=INFINITE` is specified, the marquee action loops indefinitely.

Note

If text is enclosed in a `<MARQUEE> . . . </MARQUEE>` element set, it defaults to an infinite loop action.

SCROLLAMOUNT

Specifies the number of pixels between each successive draw of the marquee text-the amount for the text to move between each draw.

SCROLLDELAY

`SCROLLDELAY` specifies the number of milliseconds between each successive draw of the marquee text; it controls the speed at which the text draw takes place.

The following marquee would be extremely fast:

```
<MARQUEE SCROLLDELAY=1 SCROLLAMOUNT=75>Hello.</MARQUEE>
```

VSPACE

This attribute is the same as that for `` (images). It specifies the number of pixels of free space at the top and bottom edges of the marquee so that the text that flows around it doesn't push up against the sides.

Note

If you want to set the `` to be displayed in the marquee, the `<MARQUEE>` definition should be enclosed inside the `<MARQUEE>`, as in this example:
`<MARQUEE>Hello!</MARQUEE>`

WIDTH

This specifies the width of the marquee, in pixels (`WIDTH=n`) or as a percentage of the screen height (`WIDTH=n%`).

<NOBR> . . . </NOBR>

The `<NOBR>` (no break) element specifies that all the text between the start and end of the `<NOBR>` elements cannot have line breaks inserted. Although `<NOBR>` may be essential for those character sequences that you don't want to be broken, it should be used carefully; long text strings inside `<NOBR>` elements can look rather odd, especially if the user adjusts the page size

by altering the window size.

Note

The `<NOBR>` element is supported only by Netscape and Internet Explorer.

`<P>...</P>`

The paragraph element indicates a paragraph of text. No specification has ever attempted to define exactly the indentation of paragraph blocks, and this may be a function of other elements, style sheets, and so on.

Typically, paragraphs should be surrounded by a vertical space of between one and one-and-a-half lines. With some browsers, the first line in a paragraph may be indented.

```
<H1>The Paragraph element</H1>
<P>The paragraph element is used to denote paragraph
blocks.</P>
<P>This would be the second paragraph.</P>
```

Included in the HTML 3.2 specification is the capability to align paragraphs.

Basically, the `ALIGN=left|center|right` attribute and values have been added to the `<P>` element.

In the following example, all text within the paragraph will be aligned to the left side of the page layout. This setting is equal to the default `<P>` element:

```
<P ALIGN=LEFT> ... </P>
```

In this example, all text within the paragraph is aligned to the center of the page. (See also `<CENTER>...</CENTER>.`)

```
<P ALIGN=CENTER> ... </P>
```

In this example, all text is aligned to the right side of the page.

```
<P ALIGN=RIGHT> ... </P>
```

Note

Internet Explorer supports only the use of the left and center values, whereas Mosaic and Netscape support the use of all three values.

`<PLAINTEXT>`

The `<PLAINTEXT>` element can be used to represent formatted text. As such, it is similar to the `<XMP>` and `<LISTING>` element. However, the `<PLAINTEXT>` element should be an open element, with no closing element. Only Netscape supports this element according to any HTML specification. Internet Explorer and Mosaic will allow the use of a `</PLAINTEXT>` closing element. Netscape treats the closing element literally and displays it.

Typically, it renders as fixed-width font with white space separating it from other text.

```
I live<PLAINTEXT>in the rainiest part of the world.
```

renders as

```
I live
in the rainiest part of the world.
```

Anything following the opening `<PLAINTEXT>` element should be treated as text. Only

Netscape behaves like this. Internet Explorer and Mosaic allow the use of a closing `</PLAINTEXT>` element, allowing discrete blocks of `<PLAINTEXT>` formatted text to be displayed.

`<PRE> . . . </PRE>`

The preformatted text element presents blocks of text in fixed-width font and is suitable for text that has been formatted on-screen or in a monospaced font.

The `<PRE>` element may be used with the optional `WIDTH` attribute, which is an HTML Level 1 feature. The `WIDTH` attribute specifies the maximum number of characters for a line and allows the browser to determine which of its available fonts to use and how to indent the text (if at all). If the `WIDTH` attribute is not present, a width of 80 characters is assumed. Where the `WIDTH` attribute is supported, widths of 40, 80, and 132 characters should be presented optimally, with other widths being rounded up.

Within preformatted text, any line breaks within the text are rendered as a move to the beginning of the next line. The `<P>` element should not be used, but if it is found, it should be rendered as a move to the beginning of the next line. It is possible to use anchor elements, and character highlighting elements are allowed. Elements that define paragraph formatting (headings, address, and so on) must not be used. The horizontal tab character (encoded in US-ASCII and ISO-8859-1 as decimal 9) represents a special formatting case. It should be interpreted as the smallest positive nonzero number of spaces that will leave the number of characters so far on the line as a multiple of 8. (However, despite being allowed, its use is not recommended.)

Note

It is at the discretion of individual browsers how to render preformatted text. Where "beginning of a new line" is implied, the browser can render that new line indented if it sees fit.

For example,

```
<PRE WIDTH="80">
This is an example of preformatted text.
</PRE>
```

Within a preformatted text element, the constraint that the rendering must be on a fixed horizontal character pitch may limit or prevent the capability of the browser to render highlighting elements specially.

`<WBR>`

The `<WBR>` element (word break) is for the very rare case when a `<NOBR>` section requires an exact break. Also, it can be used any time the browser can be helped by telling it where a word is allowed to be broken. The `<WBR>` element does not force a line break (`
` does that); it simply lets the browser know where a line break is allowed to be inserted if needed.

Note

`<WBR>` is supported only by Netscape and the Internet Explorer.

`<XMP> . . . </XMP>`

The `<XMP>` element can be used to present blocks of text in fixed-width font and is suitable for text that has been formatted on-screen. As such, it is similar to the `<PRE>` and `<LISTING>` elements but has a different syntax.

Typically, it renders as fixed-width font with white space separating it from other text. It should be rendered so that 80 characters fit on the line. For example,

```
The <XMP>Netscape Navigator</XMP>supports colored tables.
```

renders as

The
Netscape Navigator
doesn't support colored tables.

Note

The Internet Explorer translates any special characters included within <XMP> elements. If characters such as < , > , and so on are used, they are translated to < and > . Netscape and Mosaic treat the text contained within the elements literally.

Character Data

Within an HTML document, any characters between the HTML elements represent text. An HTML document (including elements and text) is encoded by means of a special character set described by the `charset` parameter as specified in the `text/html` MIME type. Essentially, this is restricted to a character set known as US-ASCII (or ISO-8859-1), which encodes the set of characters known as *Latin Alphabet No 1* (commonly abbreviated to Latin-1). This covers the characters from most Western European languages. It also covers 25 control characters, a soft hyphen indicator, 93 graphical characters, and 8 unassigned characters.

It should be noted that non-breaking space and soft hyphen indicator characters are not recognized and interpreted by all browsers; because of this, their use is discouraged.

There are 58 character positions occupied by control characters. See "Control Characters" for details on the interpretation of control characters.

Because certain special characters are subject to interpretation and special processing, information providers and browser implementers should follow the guidelines in the "Special Characters" section.

In addition, HTML provides character-entity references and numerical character references to facilitate the entry and interpretation of characters by name and by numerical position.

Because certain characters are interpreted as markup, they must be represented by entity references as described in "Character and/or Numerical References."

Character Entity References

Many of the Latin-1 set of printing characters may be represented within the text of an HTML document by a character entity.

It may be beneficial to use character entity references instead of directly typing the required characters as described in the numerical entity references; this enables you to compensate for keyboards that don't contain the required characters (such as characters common in many European languages) and for characters that may be recognized as SGML coding.

A character entity is represented in an HTML document as an SGML entity whose name is defined in the HTML DTD. The HTML DTD includes a character entity for each of the SGML markup characters and for each of the printing characters in the upper half of Latin-1, so you can reference them by name if it is inconvenient to enter them directly:

the ampersand (& ;), double quotation marks (" ;), lesser (< ;) and greater (> ;) characters

Kurt Göl;del was a famous logician and mathematician.

Note

To ensure that a string of characters is not interpreted as markup, represent

all occurrences of <, >, and & by character or entity references.

Table B.2 contains the possible numeric and character entities for the ISO-Latin-1 (ISO8859-1) character set. Where possible, the character is shown.

Note

Not all browsers can display all characters, and some browsers may even display characters different from those that appear in the table. Newer browsers seem to have a better track record for handling character entities, but be sure to test your HTML files extensively with multiple browsers if you intend to use these entities.

Table B.2. ISO-Latin-1 character set.

<i>Character</i>	<i>Numeric Entity</i>	<i>Hex Value</i>	<i>Character Entity (if any)</i>	<i>Description</i>
	�-	00-08		Unused
			09		Horizontal tab
	
	0A		Line feed
	-	0B-1F		Unused
	 	20		Space
!	!	21		Exclamation mark
"	"	22	"	Quotation mark
#	#	23		Number sign
\$	$	24		Dollar sign
%	%	25		Percent sign
&	&	26	&	Ampersand
'	'	27		Apostrophe
((28		Left parenthesis
))	29		Right parenthesis
*	*	2A		Asterisk
+	+	2B		Plus sign
,	,	2C		Comma
-	-	2D		Hyphen
.	.	2E		Period (fullstop)
/	/	2F		Solidus (slash)
0-9	0-9	30-39		Digits 0-9
:	:	3A		Colon
;	;	3B		Semicolon
<	<	3C	<	Less than
=	=	3D		Equal sign
>	>	3E	>	Greater than
?	?	3F		Question mark
@	@	40		Commercial at
A-Z	A-Z	41-5A		Letters A-Z
[[5B		Left square bracket
\	\	5C		Reverse solidus (backslash)
]]	5D		Right square bracket
^	^	5E		Caret
-	_	5F		Horizontal bar
`	`	60		Grave accent
a-z	a-z	61-7A		Letters a-z
{	{	7B		Left curly brace

	|	7C		Vertical bar
}	}	7D		Right curly brace
~	~	7E		Tilde
	- 	7F-A0		Unused
¡	¡	A1		Inverted Exclamation point
¢	¢	A2		Cent sign
£	£	A3		Pound sterling
¤	¤	A4		General currency sign
¥	¥	A5		Yen sign
	¦	A6		Broken vertical bar
§	§	A7		Section sign
¨	¨	A8		Umlaut (dieresis)
∞	©	A9	© (NHTML)	Copyright
^a	ª	AA		Feminine ordinal
<	«	AB		Left angle quotation, guillemot left
¬	¬	AC		Not sign
-	­	AD		Soft hyphen
®	®	AE	® (HHTML)	Registered trademark
ˉ	¯	AF		Macron accent
°	°	B0		Degree sign
±	±	B1		Plus or minus
²	²	B2		Superscript two
³	³	B3		Superscript three
´	´	B4		Acute accent
μ	µ	B5		Micro sign
¶	¶	B6		Paragraph sign
·	·	B7		Middle dot
¸	¸	B8		Cedilla
¹	¹	B9		Superscript one
º	º	BA		Masculine ordinal
>	»	BB		Right angle quotation, guillemot right
1/4	¼	BC		Fraction one-fourth
1/2	½	BD		Fraction one-half
3/4	¾	BE		Fraction three-fourths
¿	¿	BF		Inverted question mark
À	À	C0	À	Capital A, grave accent
Á	Á	C1	Á	Capital A, acute accent
Â	Â	C2	Â	Capital A, circumflex accent
Ã	Ã	C3	Ã	Capital A, tilde
Ä	Ä	C4	Ä	Capital A, dieresis or umlaut mark
Å	Å	C5	Å	Capital A, ring
Æ	Æ	C6	Æ	Capital AE diphthong (ligature)
Ç	Ç	C7	Ç	Capital C, cedilla
È	È	C8	È	Capital E, grave accent
É	É	C9	É	Capital E, acute accent

Ê	Ê	CA	Ê	Capital E, circumflex accent
Ë	Ë	CB	Ë	Capital E, dieresis or umlaut mark
Ì	Ì	CC	Ì	Capital I, grave accent
Í	Í	CD	Í	Capital I, acute accent
Î	Î	CE	Î	Capital I, circumflex accent
Ï	Ï	CF	Ï	Capital I, dieresis or umlaut mark
q	Ð	D0	Ð	Capital Eth, Icelandic
Ñ	Ñ	D1	Ñ	Capital N, tilde
Ò	Ò	D2	Ò	Capital O, grave accent
Ó	Ó	D3	Ó	Capital O, acute accent
Ô	Ô	D4	Ô	Capital O, circumflex accent
Õ	Õ	D5	Õ	Capital O, tilde
Ö	Ö	D6	Ö	Capital O, dieresis, or umlaut mark
r	×	D7		Multiply sign
o	Ø	D8	Ø	Capital O, slash
Û	Ù	D9	Ù	Capital U, grave accent
Ú	Ú	DA	Ú	Capital U, acute accent
Û	Û	DB	Û	Capital U, circumflex accent
Ü	Ü	DC	Ü	Capital U, dieresis or umlaut mark
s	Ý	DD	Ý	Capital Y, acute accent
	Þ	DE	Þ	Capital THORN, Icelandic
	ß	DF	ß	Small sharp s, German (sz ligature)
à	à	E0	à	Small a, grave accent
á	á	E1	á	Small a, acute accent
â	â	E2	â	Small a, circumflex accent
ã	ã	E3	ã	Small a, tilde
ä	ä	E4	&aauml;	Small a, dieresis or umlaut mark
å	å	E5	å	Small a, ring
æ	æ	E6	æ	Small ae diphthong (ligature)
ç	ç	E7	ç	Small c, cedilla
è	è	E8	è	Small e, grave accent
é	é	E9	é	Small e, acute accent
ê	ê	EA	ê	Small e, circumflex accent
ë	ë	EB	ë	Small e, dieresis or umlaut mark
ì	ì	EC	ì	Small i, grave accent
í	í	ED	í	Small i, acute accent
î	î	EE	î	Small i, circumflex accent
ï	ï	EF	ï	Small i, dieresis or umlaut

				mark
u	ð	F0	ð	Small eth, Icelandic
ñ	ñ	F1	ñ	Small n, tilde
ò	ò	F2	ò	Small o, grave accent
ó	ó	F3	ó	Small o, acute accent
ô	ô	F4	ô	Small o, circumflex accent
õ	õ	F5	õ	Small o, tilde
ö	ö	F6	ö	Small o, dieresis or umlaut mark
^	÷	F7		Division sign
h	ø	F8	ø	Small o, slash
ù	ù	F9	ù	Small u, grave accent
ú	ú	FA	ú	Small u, acute accent
û	û	FB	û	Small u, circumflex accent
ü	ü	FC	ü	Small u, dieresis or umlaut mark
v	ý	FD	ý	Small y, acute accent
	þ	FE	þ	Small thorn, Icelandic
ÿ	ÿ	FF	ÿ	Small y, dieresis or umlaut mark

Control Characters

Control characters are non-printable characters that typically are used for communication and device control, as format effectors, and as information separators.

In SGML applications, the use of control characters is limited in order to maximize the chance of successful interchange over heterogenous networks and operating systems. In HTML, only three control characters are used: horizontal tab (HT, encoded as 9 decimal in US-ASCII and ISO-8859-1), carriage return, and line feed.

A horizontal tab is interpreted as a word space in all contexts except preformatted text. Within preformatted text, the tab should be interpreted to shift the horizontal column position to the next position that is a multiple of 8 on the same line; that is, $col := ((col+8) \text{ div } 8) * 8$ (where div is integer division).

Carriage returns and line feeds conventionally are used to represent the end of a line. For Internet Media Types defined as `text/*`, the sequence CR/LF is used to represent the end of a line. In practice, `text/html` documents frequently are represented and transmitted using an end-of-line convention that depends on the conventions of the source of the document; frequently, that representation consists of CR only, LF only, or a CR/LF combination. In HTML, the end of line in any of its variations is interpreted as a word space in all contexts except preformatted text. Within preformatted text, HTML interpreting agents should expect to treat any of the three common representations of end of line as starting a new line.

Numeric Character References

In addition to any mechanism by which characters may be represented by the encoding of the HTML document, it is possible to explicitly reference the printing characters of the Latin-1 character encoding using a numeric character reference.

There are two principal cases for using a numeric character reference. First, some keyboards may not provide the necessary characters (such as those that use accents, cedillas, dieresis marks, and so on) commonly used in European languages. Second, some characters would be interpreted as SGML coding (for example, the ampersand (&), quotation marks (" "), less than

sign (<), and greater than sign (>) characters) and therefore should be referred to by numerical references.

Numeric character references are represented in an HTML document as SGML entities in which the name is a number sign (#) followed by a numeral from 32-126 and 161-255. The HTML DTD includes a numeric character for each of the printing characters of the Latin-1 encoding, so you can reference them by number if it is inconvenient to enter them directly: the ampersand (&), quotation marks ("), lesser (<) and greater (>) characters.

The following entity names are used in HTML, always prefixed by ampersand (&) and followed by a semicolon (;) (see Table B.1).

Special Characters

Certain characters have special meanings in HTML documents. There are two printing characters that may be interpreted by an HTML application to have an effect of the format of the text.

Space

This is interpreted as a single word space (the section of a paragraph of text where the text can be broken if necessary—for example, where lines can be broken for text wrapping) except when it is used within <PRE> . . . </PRE> elements. Within preformatted text elements, a space is interpreted as a nonbreaking space.

Hyphen

This is interpreted as a hyphen glyph in all contexts.

The following entity names are used in HTML, always prefixed by ampersand (&) and followed by a semicolon (;). They represent particular graphics characters that have special meanings in places in the markup or may not be part of the character set available to the writer.

<i>Glyph</i>	<i>Name</i>	<i>Syntax</i>	<i>Description</i>
<	lt	<	Less than sign
>	gt	>	Greater than sign
&	amp	&	Ampersand
"	quot	"	Quotation mark

Document Sound

Note
Two different elements now exist for employing inline sound directly in an HTML document. The first is BGSOUND; this element currently is supported only by Microsoft Internet Explorer. The other is SOUND, which currently is supported only by NCSA Mosaic. Mosaic also supports a limited version of Microsoft's BGSOUND element. Netscape can support inline sound via the plugin mechanism. See <EMBED> for more details.

The BGSOUND element allows you to create pages that play sound clips or background soundtracks while the page is being viewed. Sounds can be samples (.WAV or .AU) or MIDI (.MID) format.

<BGSOUND>

The HTML used to insert a background sound into a page is

```
<BGSOUND SRC="start.wav">
```

The **BGSOUND** element accepts the following attributes.

SRC

Specifies the address of a sound to be played.

LOOP=*n*

Specifies how many times a sound loops when activated. If *n*=-1 or **LOOP=INFINITE** is specified, the sound loops indefinitely.

Mosaic supports use of the **SOUND** element for playing inline sound. This element allows the playing of *.WAV files in pages.

Note
The SOUND element is supported only by Mosaic.

The syntax is

```
<SOUND SRC="filename.wav">
```

The **<SOUND>** element supports the following attributes:

LOOP=infinite and **DELAY=sec**

LOOP=infinite plays the sound sample continuously while the page is being viewed.

DELAY=sec delays playing of the sound file for *sec* seconds after the page and sound file finish loading.

Note
Although Mosaic supports the use of the BGSOUND element (for *.WAV file), it does not play inline *.MID MIDI files without launching an external application as defined in the Helper Application setup.

Dynamic Documents

Recent advances in browser technology have been pushing the idea of active content. To this end, you should be aware of a few HTML methods:

Server push This mechanism generally has been used for providing animation within Web pages, whereby the Web server serves the page that the browser has requested and keeps the client (browser) to server connection open and repeatedly sends chunks of data as long as the connection is kept open. To be able to take advantage of such a mechanism requires an in-depth knowledge of MIME types, the HTTP transport protocol, and CGI scripting or programming. It therefore is recommended only for programmers.

Client pull As seen in the discussion of the **<META>** element, this method provides a useful automatic redirection mechanism for serving Web pages. The server serves the browser the requested page (which contains metainformation) which makes the browser believe that it has received certain HTTP response headers, which typically would be used to make the browser retrieve a different document. For more details, see the **<META>** element.

Server Push

Server push allows for dynamic document updating via a server-to-client connection that is kept open. This method (as opposed to client pull) is totally controlled by the server, but the perpetual open connection occupies valuable server resources. Its main advantage over client pull is that you can use server push to replace a single inline image in a page repeatedly. All that is needed is that the **SRC** attribute of the image to be updated points to a URL that continually pushes

image data through the open HTTP connection.

The exact server push mechanism is technically complex and is outside the scope of this reference. This section presents a brief outline of the method. Those interested in utilizing server push in CGI scripts or Web server-based executable applications should visit the Netscape Web site (<http://home.netscape.com/>) for more information. It should be noted that only Netscape supports the use of server push.

When a Web server receives a request for an HTML document to be retrieved, it typically sends a single set of data (the actual HTML document). MIME possesses a facility whereby many pieces of data can be sent encapsulated in a single message by using the MIME type `multipart/mixed` where the message is split into separate data sections, each provided with their own MIME type (given in the content header) so that the browser can distinguish between the different data in the different sections of the message. Server push uses a variation on this MIME type, called `multipart/x-mixed-replace` (the `x-` represents the fact that the MIME type is experimental and has not achieved standardized use). It is by virtue of the `replace` section that certain sections of the message can be replaced. Essentially, the server does not push down the entire message at once. It sends down sections (data chunks) of the message when it sees fit (or as controlled by the server push script or application). When the browser sees a separator (sent down in the `multipart/x-mixed-replace` message), it just sits and waits for the next data object to be sent, which it then uses to replace the data previously sent by the server.

Forms

Perhaps the biggest advance the HTML 2.0 specification made over its predecessors was the inclusion of elements that allowed for users to input information. These elements are the `<FORM>` elements. They provide for the inclusion of objects like text boxes, choice lists, and so on, and have proved invaluable for recent HTML applications-particularly, search engines, database query entry, and so on.

It should be noted that although these HTML elements can be used to easily define the presentation of the form to the user, the real value behind any form is in what it does with the information that is entered. For a form to do anything more than send a straight text dump of the form data (including control characters) to an e-mail address, the form data will need to be passed to some kind of CGI script or server-based executable for processing.

The following elements are used to create forms:

<code><FORM>...</FORM></code>	A form within a document
<code><INPUT ...>...</INPUT></code>	One input field
<code><OPTION></code>	One option within a <code>Select</code> element
<code><SELECT>...<SELECT></code>	A selection from a finite set of options
<code><TEXTAREA ...>...</TEXTAREA></code>	A multi-line input field

Each variable field is defined by an `INPUT`, `TEXTAREA`, or `OPTION` element and must have a `NAME` attribute to identify its value in the data returned when the form is submitted.

A very simple form for eliciting user response follows, and its output is shown in Figure B.1.

```
<H1 ALIGN="center">Comment Form</H1>
<FORM METHOD="POST"
ACTION="http://www.htmlib.com/formscript.cgi">
<CENTER>
Your name: <INPUT NAME="name" size="20">
Your e-mail address: <INPUT NAME="email" size="20">
<P>I think the HTML Reference is:
  <SELECT NAME="Choice">
    <OPTION>Outstanding
    <OPTION>Very good
```

```
<OPTION>Good
<OPTION>Average
<OPTION>Below Average
<OPTION>Awful
<OPTION SELECTED>My response would be "indecent" under
the CDA Act.
</SELECT>
<P>If you have any further comments, please enter them
here:<BR>
  <TEXTAREA NAME="Comments" ROWS="10" COLS="40"
  WRAP="Virtual">
  </TEXTAREA>
<P><INPUT TYPE=SUBMIT> <INPUT TYPE=RESET>
</CENTER>
</FORM>
```

Different platforms will have different native systems for navigating within the input fields of a form. (For example, Windows users can use the Tab key to move from one field to the next through the order that the fields appear within the form.) Different browsers also may display different text on any buttons included in the form. For example, Netscape defaults to displaying Submit Query for a button specified by <INPUT TYPE=SUBMIT>, whereas the Internet Explorer and Mosaic display just Submit on such a button.

HTTP File Upload

It is possible to write forms that ask for files as input, rather than data input by input boxes and other simple elements such as check boxes and radio buttons.

An example of such a form follows:

```
<FORM ENCTYPE="multipart/form-data" ACTION="_URL_"
METHOD=POST>
Send this file: <INPUT NAME="userfile" TYPE="file">
<INPUT TYPE="submit" VALUE="Send File">
</FORM>
```

Note

This method of file upload is Netscape specific and is essentially adoption of another IETF Internet Draft by the Netscape authors. The Internet Draft in question, "Form based file upload in HTML," details adding the FILE option to the TYPE attribute of the INPUT element, allowing an ACCEPT attribute for the INPUT element (which would be a list of MIME types—essentially detailing what files are allowed to be uploaded as the contents of the form) and allowing the ENCTYPE of a form to be multipart/form-data. This MIME type essentially wraps the form data (including that presented in any other input fields) as a data stream, with discrete boundaries between the information sections. For a more detailed description, readers should check the HTTP file upload specification.

The display method is largely at the discretion of the browsers that support this method. Netscape (Windows versions) displays a Browse button beside the input box, which brings up the standard Open/Save dialog box, allowing the choice of any local file for upload.

<FORM> . . . </FORM>

The <FORM> element is used to delimit a data input form. There can be several forms in a single document, but the <FORM> element cannot be nested. (A form can't contain another form.)

```
<FORM ACTION="_URL_" METHOD="GET|POST" ENCTYPE="MIME
```

type">

The **ACTION** attribute is a URL specifying the location to which the contents of the form data fields are submitted to elicit a response. As mentioned before, this could be simply a direction to an e-mail address, but generally it is used to point toward some kind of server-based CGI script/application that handles the forwarding of form data. If the **ACTION** attribute is missing, the URL of the document itself is assumed. The way data is submitted varies with the access protocol of the URL to which the form data is sent and with the values of the **METHOD** and **ENCTYPE** attributes.

Generally, the **METHOD** attribute specifies a method of accessing the URL specified in the **ACTION** attribute. Generally, the method is **GET** or **POST**. The **GET** method is ideal for form submission where the use of the form data does not require external processing. With database searches, for example, there is no lasting effect caused by the query of the form (the query runs its search through the database and reports the results). However, when the form is used to provide information that updates a database, the **POST** method should be used, with the **ACTION** attribute pointing to a CGI script that executes the form data processing.

The **ENCTYPE** specifies the media type used to encode the form data. The default **ENCTYPE** is the MIME type `application/x-www-form-urlencoded`.

<INPUT>

The **<INPUT>** element represents a field whose contents may be edited or activated by the user.

Attributes of the **<INPUT>** element follow.

ALIGN

To be used with the **TYPE=IMAGE** setting, this attribute specifies the alignment of the image. It takes the same values as the **ALIGN** in the **** element.

CHECKED

To be used with a **TYPE=CHECKBOX** or **TYPE=RADIO** setting, this indicates that the check box or radio button is selected.

MAXLENGTH

To be used with **TYPE=TEXT** setting, this indicates the maximum number of characters that can be entered into a text field. This can be greater than specified by the **SIZE** attribute, in which case the field scrolls appropriately. The default number of characters is unlimited.

NAME

Represents the name that will be used for the data when transferring the form's contents. The **NAME** attribute is required for most input types and normally is used to provide a unique identifier for a field or a logically related group of fields.

SIZE

Specifies the size or precision of the field according to its type. For example, to specify a field with a visible width of 24 characters, use this code:

```
INPUT TYPE=text SIZE="24"
```

SRC

To be used with the **TYPE=IMAGE**, represents a URL specifying the desired image.

TYPE

Defines the type of data the field accepts. Defaults to free text. Several types of fields can be defined with the type attribute:

- **BUTTON** Can be used to embed buttons directly into HTML documents that add functionality when used with VBScript. The **NAME** attribute is used to give the button a unique name, which can be used to set its function in the script. The **VALUE** attribute specifies the text displayed on the button in the document.
- **CHECKBOX** Used for simple Boolean attributes (where a field will be chosen, or not) or for attributes that can take multiple values at the same time. The latter is represented by a number of check box fields, each of which has the same name. Each selected check box generates a separate name/value pair in the submitted data, even if this results in duplicate names. The default value for check boxes is **on**. It requires the **NAME** and **VALUE** attributes, and optional attributes are **CHECKED**.
- **FILE** Netscape now supports a **FILE** option to the **TYPE** attribute of the **INPUT** element, allowing an **ACCEPT** attribute for the **INPUT** element (which is a list of media types or type patterns allowed for the input) and allowing the **ENCTYPE** of a form to be **multipart/form-data**.
This allows the inclusion of files with form information, which could prove invaluable for companies providing technical support or for service providers requesting data files.
- **HIDDEN** With this input type, no field is presented to the user, but the content of the field is sent with the submitted form. This value may be used to transmit state information about client/server interaction.
- **IMAGE** An image field on which you can click with a pointing device, causing the form to be submitted immediately. The coordinates of the selected point are measured in pixel units from the upper left corner of the image and are returned (along with the other contents of the form) in two name/value pairs. The x coordinate is submitted under the name of the field with **.x** appended, and the y coordinate is submitted under the name of the field with **.y** appended. The **NAME** attribute is required. The image itself is specified by the **SRC** attribute, exactly as for the **<IMAGE>** element.

Note

In a future version of the HTML specification, the **IMAGE** functionality may be folded into an enhanced **SUBMIT** field.

- **PASSWORD** **PASSWORD** is the same as the **TEXT** attribute, except that text is not displayed as it is entered.
- **RADIO** Used for attributes that accept a single value from a set of alternatives. Each radio button field in the group should be given the same name. Only the selected radio button in the group generates a name/value pair in the submitted data. Radio buttons require an explicit **VALUE** and **NAME** attribute. **CHECKED** is an optional attribute and can be used to specify which options are selected for initial form display.
- **RESET** **RESET** is a button that resets the form's fields to their specified initial values. The label to be displayed on the button may be specified just as for the **SUBMIT** button.
- **SUBMIT** **SUBMIT** is a button that submits the form. You can use the **VALUE** attribute to provide a non-editable label to be displayed on the button. The default label is browser-specific. If a Submit button is clicked in order to submit the form and that button has a **NAME** attribute specified, that button contributes a name/value pair to the submitted data. Otherwise, a Submit button makes no contribution to the submitted data.

- **TEXT** Used for single-line text-entry fields. It should be used with the **SIZE** and **MAXLENGTH** attributes to set the maximum amount of text that can be entered. For textual input that requires multiple lines, use the **<TEXTAREA>** element for text fields that can accept multiple lines. Explicit **VALUE** and **NAME** attributes are also required.
- **TEXTAREA** Used for multiple-line text-entry fields. Use with the **SIZE** and **MAXLENGTH** attributes.

VALUE

When used with **TYPE= . . .** attributes, this attribute sets the initial displayed value of the field if it displays a textual or numerical value. If the **TYPE= . . .** attribute is one that only allows Boolean values (chosen or not chosen), this specifies the value to be returned when the field is selected.

<OPTION>

The **<OPTION>** element can occur only within a **<SELECT>** element. It represents one choice and can take these attributes:

1. **SELECTED** Indicates that this option initially is selected.
2. **VALUE** When present, indicates the value to be returned if this option is chosen. The returned value defaults to the contents of the **<OPTION>** element. The contents of the **<OPTION>** element are presented to the user to represent the option. It is used as a returned value if the **VALUE** attribute is not present.

<SELECT . . .> . . . </SELECT>

The **<SELECT>** element allows the user to choose one of a set of alternatives described by textual labels. Every alternative is represented by the **<OPTION>** element.

Attributes used with **<SELECT>** follow:

1. **MULTIPLE** Needed when users are allowed to make several selections-for example, **<SELECT MULTIPLE>**.
2. **NAME** Specifies the name that will be submitted as a name/value pair.
3. **SIZE** Specifies the number of visible items. If this is greater than one, then the resulting form control will be a list.

The **<SELECT>** element typically is rendered as a pull-down or pop-up list, as in this example:

```
<SELECT NAME="Choice">
  <OPTION>Outstanding
  <OPTION>Very good
  <OPTION>Good
  <OPTION>Average
  <OPTION>Below Average
  <OPTION>Awful
  <OPTION SELECTED>My response would be "indecent" under
the CDA Act.
</SELECT>
```

<TEXTAREA> . . . </TEXTAREA>

The **<TEXTAREA>** element lets users enter more than one line of text.

Any text included up to the end element (**</TEXTAREA>**) is used to initialize the field's value. This end element always is required even if the field initially is blank. When submitting a form,

lines in a TEXTAREA should be terminated using CR/LF.

In a typical rendering, the ROWS and COLS attributes determine the visible dimension of the field in characters. The field is rendered in a fixed-width font. Browsers should allow text to extend beyond these limits by scrolling as needed.

Recent versions of Netscape (from version 2.0) have introduced the WRAP attribute in the <TEXTAREA> element: Now it is possible to specify how to handle word-wrapping display in text-input areas in forms.

<TEXTAREA WRAP=OFF>	The default setting. Wrapping doesn't happen. Lines are sent exactly as typed.
<TEXTAREA WRAP=VIRTUAL>	The display word wraps, but long lines are sent as one line without new lines.
<TEXTAREA WRAP=PHYSICAL>	The display word wraps, and the text is transmitted at all wrap points.

Note

Word wrapping in a TEXTAREA text box is supported by Netscape only.

Advanced Page Formatting

Note

The use of frames currently is supported only by recent versions of Netscape (from version 2.0) and Internet Explorer (3.0 and above).

Frames allow the browser display window to be subdivided into separate sections, each of which can be updated, or have new documents loaded into it separately from the remaining frame sections. As such, a frame-based layout can be especially useful for HTML applications where some information is required across a whole range of pages (such as a table of contents or title graphics, for example).

Frames are generated by three elements: <FRAMESET>, <FRAME>, and <NOFRAMES>.

Frame Document

A frame document has a basic structure very much like a normal HTML document, except the BODY container is replaced by a FRAMESET container that describes the sub-HTML documents or frames that will make up the page:

```
<HTML>
<HEAD>
</HEAD>
<FRAMESET>

</FRAMESET>
</HTML>
```

No HTML that normally would be included within the <BODY> section of an HTML document should be included within the <FRAMESET> . . . </FRAMESET> elements.

Frame Syntax

<FRAMESET>

This is the main container for a frame. It has two attributes: ROWS and COLS. The <FRAMESET> element has a matching end element, and within the <FRAMESET> you can have other nested <FRAMESET>, <FRAME>, or <NOFRAMES> elements.

ROWS="row_height_value_list"

This takes a list of values, separated by commas. They can represent absolute pixel, percentage, or relative scaling values. The total set by the values given in the ROWS attribute should not exceed 100 percent (the total rows are extended across the whole available browser display window).

If any of the values are single numerical values, these are considered to be absolute pixel values. It is not recommended to fix a frame set by using a complete set of pixel values, because browsers use a variety of screen resolutions when viewing documents, so the layout may become distorted. Percentage values can be given for this attribute. If the total percentage values given exceed 100 percent, all values will be scaled down by the browser so that the total is 100 percent. The remaining value option is to use an asterisk (*) character. This tells the browser that the frame is a relative size frame and should be displayed accordingly. Numerical values can be used with the * character to scale the relative frame sections within the browser window.

To specify a three-framed vertical layout where the first section uses 20 percent of the display window, the second section uses 100 pixels, and the third section uses the remaining screen, use this code:

```
<FRAMESET ROWS="20%, 100, *>
```

To split the layout into two vertical frames (the first using a quarter of the display window and the second using three-quarters of the window), use this:

```
<FRAMESET ROWS="25%, 75%>
```

This is exactly the same as using `<FRAMESET ROWS="*, 3*">`.

COLS="column_width_list"

The COLS attribute takes as its value a comma-separated list of values that is of the exact same syntax as the list described for the ROWS attribute.

The `<FRAMESET>` element can be nested. In this way, frame sections can be set up where the display window can be split into horizontal or vertical sections, with any of these further sub-divided by nested `<FRAMESET>` elements.

<FRAME>

This element defines a single frame in a frameset. It has eight possible attributes: SRC, NAME, MARGINWIDTH, MARGINHEIGHT, SCROLLING, NORESIZE, FRAMEBORDER, and FRAMESPACING. The `<FRAME>` element is not a container, so it has no matching end tag.

SRC="url"

This attribute is used to specify the HTML document that will be used as the display in the particular frame section of the frameset.

NAME="frame_name"

The NAME attribute is used to assign a name to a frame so it can be targeted by links in other documents by using ``. (These usually would be from other documents in the same frameset.) The NAME attribute is optional; by default, all windows are unnamed.

Names must begin with an alphanumeric character. Several reserved names have been defined, which start with an underscore:

<code>_blank</code>	Always load this link into a new, unnamed window.
<code>_self</code>	Always load this link over the document that originated the link.
<code>_parent</code>	Always load this link over the parent frame (becomes <code>_self</code> if the frame has no parent or is the parent frame).
<code>_top</code>	Always load this link at the top level (becomes <code>_self</code> if the frame is the top frame).

Note

Although these are reserved names for the `NAME` attribute of the `<FRAME>` element, they should only be referred to using an anchor target. They should be used to target specific windows, allowing smoother transition between framed documents and between framed and non-framed documents (for example, when providing a link to documents on a foreign server that may not be framed documents). Although Internet Explorer supports the naming of frames for document navigation and hyperlinking, it doesn't support the use of the `_blank` reserved name for opening a document in a new browser window. Also, unlike Netscape, Internet Explorer will not open a new window for a link whose `TARGET` value has not been defined by a `NAME` attribute.

MARGINWIDTH="value"

This accepts an absolute pixel value and forces indentation from the left- and right-hand side of the frame pane according to the number of pixels. It cannot be set to a value less than 1 because this would cause the contents of the frame to be displayed right up against the left-hand margin. By default, the browser chooses its own `MARGINWIDTH` when trying to produce the best possible display.

MARGINHEIGHT="value"

This is similar to the `MARGINWIDTH` attribute, but it controls the top and bottom margins.

SCROLLING="yes | no | auto"

This attribute can be used to control the appearance of any scroll bars that may appear as a result of the frame contents being too much to display in the set pane. Using `no` may be dangerous, because the HTML author cannot know the resolution/display window size of the client browser, so information may not be displayable.

NORESIZE

By default, all frames specified in a framed document can be resized by the client. Setting this flag (it requires no value) prevents the frame from being resized.

FRAMEBORDER="yes | no"

This is an Internet Explorer specific attribute, which allows control of the frame border display. With this attribute set to `no`, the borders for the specific frame are not drawn.

FRAMESPACING="value"

This attribute is Internet Explorer-specific and allows the setting of extra space around frames to give the appearance of floating frames. The `value` should be the distance required around the frame in pixels.

For example,

```
<FRAME FRAMESPACING="40" ...>
```

presents the frame with an invisible border of 55 pixels.

<NOFRAMES>

This element is provided for HTML authors who want to create alternative content for browsers that cannot display frames. This is especially useful if the author is making the very first document of the site a framed document. It should be noted that this element is not actually recognized by non-frame-capable browsers. As with any HTML, if the browser does not recognize the element, it ignores it. Non-frame-capable browsers ignore all the <FRAMESET> and <FRAME> elements, but display whatever is enclosed in the <NOFRAMES> . . . </NOFRAMES> elements, which can be any HTML at all, because that is what it recognizes. On the other hand, frame-capable browsers preferentially display what is set up by the frame elements, unless they provide any mechanism where the display of frames can be turned off (in which case they may display this alternative content).

The Main Frame Setup Document

The main document that sets up the sample frame follows:

```
<HTML>
<!--HTMLIB.HTM-->
<HEAD>
<TITLE>The HTML Reference Library</TITLE>
</HEAD>
<BASEFONT SIZE=3>

<FRAMESET ROWS="85, *, 65">
<FRAME SCROLLING="no" NAME="title" NORESIZE
SRC="title.htm">
<FRAMESET COLS="40%, 60%">
<FRAME SCROLLING="yes" NAME="toc" SRC="toc.htm">
<FRAME SCROLLING="yes" NAME="main page" SRC="main.htm">
</FRAMESET>
<FRAME SCROLLING="no" NAME="HLP buttons" NORESIZE
SRC="buttons.htm">

<NOFRAME>

</NOFRAME>
</FRAMESET>
</HTML>
```

A Line-by-Line Breakdown

```
<FRAMESET ROWS="85, *, 65">
```

This line divides the page into three regions-the top region is 85 pixels in height, the bottom region is 65 pixels in height, and the middle region occupies the rest of the browser window.

```
<FRAME SCROLLING="no" NAME="title" NORESIZE
SRC="title.htm">
```

This line sets the top region of the window (the region that is 85 pixels high) to be a non-scrolling, non-resizable region. Its name is `title` (so any other link that specifies `title` with its `TARGET` attribute is displayed in this region).

```
<FRAMESET COLS="40%, 60%">
```

This is a nested <FRAMESET> element that splits the middle region of the browser window into two sections horizontally. The left-hand section is 40 percent of the frame width, and the right-

hand section is the remaining 60 percent of the frame width. (This also could have been achieved using `<FRAMESET COLS="2*, 3*>`.)

```
<FRAME SCROLLING="yes" NAME="toc" SRC="toc.htm">
<FRAME SCROLLING="yes" NAME="main page" SRC="main.htm">
```

These two lines (as the other `<FRAME>` line) set the attributes for the two middle sections of the page. It names the regions `toc` and `main page`, respectively, and links to the two pages to be displayed in the regions.

```
</FRAMESET>
```

This line closes the subframes that were opened in the middle section of the main framed regions.

```
<FRAME SCROLLING="no" NAME="buttons" NORESIZE
SRC="buttons.htm">
```

This line defines the properties of the remaining main region of the window-the bottom region that is 65 pixels high. It defines it as a non-scrolling, non-resizable region (ideal for navigation tools).

The Title Document

Note
This document contains no mark up relevant to the use of the frames, but it is included for reasons of completeness.

This document is the title for the paged document. It resides in the top frame, which is a non-scrolling, non-resizable frame. The title therefore will always be displayed in the same place. Note that for frame subdocuments, titles are not required. The title of the site will always be taken from the main frame page.

```
<HTML>
<!--TITLE.HTM-->
<BODY>
<BASEFONT SIZE=3>
<CENTER>
<H2 ALIGN=center>Hello and Welcome to the HTML Reference
Library</H2>
<BR>
</CENTER>
</BODY>
</HTML>
```

The Contents Document

This is the Table of Contents page. It appears on the left scrolling frame region. This section has been used (in this example) for a stationary table of contents.

```
<HTML>
<!--TOC.HTM-->
<BODY>
<BASEFONT SIZE=2>
<CENTER>
Please Select a Volume<BR><BR>
<A HREF="lang.htm" TARGET="main page"><B>1) The HTML
Language</B></A><BR>
<A HREF="qr.htm" TARGET="main page"><B>2) Quick Reference
Guide</B></A><BR>
<A HREF="author.htm" TARGET="main page"><B>3) Contacting
```

```
the Author</B></A><BR>
<A HREF="new.htm" TARGET="main page"><B>4) New in this
version</B></A><BR>
</CENTER>
</BODY>
</HTML>
```

The use of the `TARGET` attribute in the anchor means that when each link is activated the document accessed will be displayed in the frame region named `main page`. Thus, any documents accessed from the table of contents appear in the framed region to the right of the table of contents.

The Main Text Document

Note
This document contains no mark up relevant to the use of the frames, but it is included for reasons of completeness.

This document is the document that appears in the right-hand framed region of the page the first time the page is accessed.

```
<HTML>
<!--MAIN.HTM-->
<BODY>
This reference, using the Internet Draft as an information
base is an on-line
reference library of currently supported HTML elements -
their syntax, and
use.<BR>
It assumes that the user has knowledge of the World Wide
Web and the various
browsers available. Information on specific browsers, or
the broader topic
of 'The World Wide Web' can be obtained by reading the
World Wide Web FAQ.<BR>
</BODY>
</HTML>
```

The Navigation Buttons Document

Note
This document contains no mark up relevant to the use of the frames, but it is included for reasons of completeness.

This document resides at the bottom of the framed document. This region is a non-scrollable, non-resizable region. As such, it is ideal for a set of navigation buttons or other tools. For the purposes of this example, the buttons are just a graphics image.

```
<HTML>
<!--BUTTONS.HTM-->
<BODY>
<CENTER>
<IMG SRC="buttons.gif"><BR>
<FONT SIZE=1>&copy; Stephen Le Hunte 1995</FONT>
</CENTER>
</BODY>
</HTML>
```

The HTML Language Document

Note

This document contains no mark up relevant to the use of the frames, but it is included for reasons of completeness.

This document is accessed by choosing the first option from the table of contents. When accessed, it is displayed in the right-hand section of the middle regions.

```
<HTML>
<!--LANG.HTM-->
<BODY>
<CENTER><B>The HTML Language</B></CENTER>
<BR>
The vast range of HTML MarkUp currently supported by
available browsers
(Web browsers, such as Netscape, Mosaic etc.) can be
divided into the
following sections. Some elements featured here may not be
supported by
all browsers. Where an element is known to be supported by
specific
browsers, the element description will be labeled as
such.<BR>
</BODY>
</HTML>
```

Inline Images

Recently, the `` element has undergone the largest enhancements of all HTML 2.0 elements on the way to newer HTML standardization. This is due to the `` element being probably the second most important markup element (behind the anchor element) because it handles all embedded graphical content in HTML documents.

The attributes commonly supported by the `IMG` element have had some recent additions to allow client-side imagemaps, embedded inline video clips, and embedded inline VRML worlds.

Formats

Netscape and Mosaic (and most other browsers) will only support use of GIF and JPG images within HTML documents. This can be extended with Netscape by embedding image formats within pages, providing the format is one that users have software to handle the installation on their system, or they have a plugin module specifically to handle that type of image (see `<EMBED>`). Also, Netscape natively supports progressive JPEG images.

Internet Explorer allows the use of GIF, JPEG, progressive JPEG images, PNG (portable network graphics) images, and BMP files, giving the author a wider variety of image formats from which to choose.

Netscape now fully supports the GIF89a format, which means that multi-image GIF files can be used to create animation sequences. Users are encouraged to seek out the GIF Construction Kit for more details and tools for the preparation of multi-image GIF files.

``

The Image element is used to incorporate inline graphics (typically icons or small graphics) into an HTML document. This element cannot be used for embedding other HTML text.

Browsers that cannot render inline images ignore the `` element unless it contains the `ALT` attribute.

The `` element, which is empty (no closing element), has these attributes:

ALIGN

The `ALIGN` attribute accepts the values `left`, `right`, `top`, `texttop`, `middle`, `absmiddle`, `baseline`, `bottom`, and `absbottom`, which specify the alignment of the image and that of the following line of text.

Note
Not all browsers support the left and right alignment of images and will render embedded images on their own paragraph space in the browser window.

These attribute values to the `ALIGN` option require some explanation. First, the values `left` and `right`: Images with those alignments are *floating* image types.

`ALIGN=left` aligns the image on the left-hand edge of the browser display window and subsequent text wraps around the right-hand side of that image.

`ALIGN=right` aligns the image on the right-hand edge of the browser display window and subsequent text wraps around the left-hand side of that image.

The use of floating images and wraparound text can cause some formatting problems. Using `<BR CLEAR=left|right|all>` is recommended to achieve the desired page formatting effect.

`ALIGN=top` allows any text following the image to align itself with the top of the tallest item in the line (the top of the image).

`ALIGN=texttop` allows any text following the image to align itself with the top of the tallest text in the line (usually but not always the same as `ALIGN=top`).

`ALIGN=middle` aligns the baseline of the current line with the middle of the image.

`ALIGN=absmiddle` aligns the middle of the current line with the middle of the image.

`ALIGN=baseline` aligns the bottom of the image with the baseline of the current line.

`ALIGN=bottom` aligns the bottom of the image with the baseline of the current line.

`ALIGN=absbottom` aligns the bottom of the image with the bottom of the current line.

ALT

This attribute allows the setting of text as an alternative to the graphic for rendering in non-graphical environments or when the user has deactivated the auto-loading of images. Alternate text should be provided by the browser whenever the graphic is not rendered.

```
<IMG SRC="triangle.gif" ALT="Warning:"> Be sure to read  
these instructions.
```

Internet Explorer uses any `ALT` text that is set as a Tool Tip that is displayed whenever the mouse pauses over an image for which the `ALT` text has been specified.

BORDER=value

This lets the document author control the thickness of the border around an image displayed.

It is useful if the image is to be a hyperlink, because `BORDER` can be set to 0 to avoid the display of the standard blue hypertext link border.

ISMAP

The `ISMAP` (is map) attribute identifies an image as an imagemap. *Imagemaps* are graphics in which certain regions are mapped to other documents. By clicking on different regions, different resources can be accessed from the same graphic.

```
<A HREF="http://machine/htbin/imagemap/sample">
<IMG SRC="sample.gif" ISMAP></A>
```

Note

To be able to employ this type of imagemap in HTML documents, the HTTP server that will be controlling document access must have the correct cgi-bin software installed to control imagemap behavior. The document must have access to an imagemap-handling script and the mapfile defining the graphic hotspots.

Recent browsers allow a simpler form of imagemap known as *client-side imagemap*s. Although this is currently a proposed extension to HTML, it is widely supported by browsers. For details, see "Client-Side Imagemaps."

LOWSRC

Using the `LOWSRC` attribute, it is possible to use two images in the same space. The syntax is

```
<IMG SRC="hiquality.gif" LOWSRC="lowquality.gif">
```

Browsers that do not recognize the `LOWSRC` attribute ignore it and simply load the image specified by the `SRC` attribute.

Browsers that support this attribute load the image called `lowquality.gif` on their first layout pass through the document. When the rest of the document is completely loaded and formatted on the page, the browser redraws the page and loads the image specified by the standard `SRC` attribute. This allows the author to specify a low resolution (or smaller file size version of the main image- perhaps a grayscale version) image to be displayed initially while the document is loading, which later is replaced by the higher quality version.

Any graphics file format that the browser supports can be used interchangeably within the `LOWSRC` and `SRC` attributes. You also can specify width and/or height values in the `IMG` element, and both the high-resolution and low-resolution versions of the image are appropriately scaled to match. However, if no width and height values have been set, the values used for the `LOWSRC` image (the dimensions of that image) are used to rescale the `SRC` image. This is to minimize page format disruption caused by the browser trying to load two different-sized images into the same page space.

```
<IMG ALIGN="left" SRC="mosaic.gif" HSPACE="20" ALT="Mosaic
logo">Mosaic,
from the <B>N</B>ational <B>C</B>entre for
<B>S</B>upercomputing
<B>A</B>pplications represents the original graphical
browser which
Netscape development was based on.
<BR CLEAR="all">
<HR>
<IMG ALIGN="right" SRC="netscape.gif" HSPACE="20"
ALT="Netscape logo">Netscape,
from <B>Netscape Communications</B>, after initial
development from Mosaic,
stormed away and became more or less the <I>de facto</I>
```



```
Web browser.  
<BR CLEAR="all">  
<HR>  
<IMG ALIGN="left" SRC="iexplore.gif" HSPACE="20"  
ALT="Internet Explorer logo">  
Internet Explorer, from <B>Microsoft</B>, exhibits  
Microsoft's serious  
intentions to enter the Web browser market and compete  
head-to-head with  
Netscape.  
<BR CLEAR="all">  
<HR>
```

SRC

The value of the SRC attribute is the URL of the image to be displayed. Its syntax is the same as that of the HREF attribute of the <A> element. SRC is the only mandatory attribute of the element. Image elements are allowed within anchors.

```
<IMG SRC ="warning.gif">Be sure to read these  
instructions.
```

The SRC attribute can accept fully qualified or partial relative URLs, or even just image names (if the image is located in the same directory as the HTML document).

VSPACE=value HSPACE=value

For the *floating* images (those displayed with an ALIGN=left|right attribute), it is likely that the author does not want the text wrapped around the image to be pressed up against the image. VSPACE controls the vertical space above and below the image, while HSPACE controls the horizontal space to the left and right of the image. Value should be a pixel value.

WIDTH=value HEIGHT=value

The WIDTH and HEIGHT attributes allow the browser to determine the text layout surrounding images before the entire image downloads, which can significantly speed up display of the document text. If the author specifies these, the viewer of the document will not have to wait for the image to be loaded over the network and its size calculated. Internet Explorer uses image-placement mechanisms, so that if the display of inline images has been turned off, the space that the images would occupy in the page is marked as if the image were there (with any ALT text being displayed in the placeholder). This allows authors to be sure that the text layout on the page will be as desired, even if the user is not displaying the images.

Client-Side Imagemaps

Before this imagemap method was implemented by browsers, using imagemaps required communication with the Web server on which the HTML documents were located in order to determine the action to be taken when an area of the image had been clicked on. This produced unnecessary server-side overheads. The client-side imagemap specification (designed by Spyglass) allows for all of the processing of the imagemap action to be done by the browser. It allows the use of imagemaps within HTML documents that are not being distributed by conventional means (from a Web server). For example, using client-side imagemaps allows imagemap functionality for HTML documents on CD-ROMs and so on.

Basically, adding the USEMAP attribute to an element indicates that the image is a client-side imagemap. The USEMAP attribute can be used with the ISMAP attribute to indicate that the image can be processed as a client-side or server-side imagemap (useful to ensure browser independence of HTML documents). The value used in the USEMAP attribute specifies the location of the map definition to use with the image, in a format similar to the HREF attribute on anchors. If the argument to USEMAP starts with a #, the map description is assumed to be in the same document as the IMG tag.

```
<IMG SRC="../../images/image.gif" USEMAP="maps.html#map1">
```

This uses the map described as map1 in maps.html as the overlay for the image file image.gif. The map definition can be included within the HTML document where the image is embedded or in a completely separate file.

The active regions of the image are described using MAP and AREA elements.

<MAP>

The map describes each region in the image and indicates the location of the document to be retrieved when the defined area is activated. The basic format for the MAP element follows:

```
<MAP NAME="name">
<AREA [SHAPE="shape"] COORDS="x,y,..." [HREF="reference"]
[NOHREF]>
</MAP>
```

The name specifies the name of the map so that it can be referenced by an element. The shape gives the shape of the specific area. Currently, the only shape defined is "RECT", but the syntax is defined to allow other region types to be added. If the SHAPE attribute is omitted, SHAPE="RECT" is assumed. The COORDS attribute gives the coordinates of the shape, using image pixels as the units. For a rectangle, the coordinates are given as "left,top,right,bottom". The rectangular region defined includes the lower-right corner specified; to specify the entire area of a 100x100 image, the coordinates are "0,0,99,99".

The NOHREF attribute indicates that clicks in this region should perform no action. An HREF attribute specifies where a click in that area should lead. Note that a relative anchor specification will be expanded using the URL of the map description as a base, instead of using the URL of the document from which the map description is referenced. If a BASE tag is present in the document containing the map description, that URL is used as the base to resolve partial URLs.

<AREA>

An arbitrary number of <AREA> elements may be specified. If two areas intersect, the one that appears first in the map definition takes precedence in the overlapping region. For example, a button bar in a document might use a 200 × 80 pixel image and appear like this:

```
<MAP NAME="buttonbar">
<AREA SHAPE="RECT" COORDS="10,10,40,70"
HREF="../../index.html">
<AREA SHAPE="RECT" COORDS="60,10,90,70"
HREF="../../download.html">
<AREA SHAPE="RECT" COORDS="110,10,140,70"
HREF="../../email.html">
<AREA SHAPE="RECT" COORDS="160,10,190,70"
HREF="../../reference.html">
</MAP>
<IMG SRC="../../images/tech/bar.gif" USEMAP="#buttonbar">
```

Note

The TARGET attribute can be used within the <AREA> element, allowing the use of client-side imagemaps within framed documents. For more information about the use of TARGET attributes, see the <FRAME> section.

Inline Video

Microsoft's Internet Explorer allows the user to embed .AVI (Audio Video Interleave) video

clips in HTML documents. This is done by adding several new attributes, notably `DYNSRC` (dynamic source) to the `` element. Using the `IMG` element for this purpose makes it possible to add video clips to page, but also have non-video-enabled browsers display still images in their place.

Note

In future versions of Internet Explorer, proprietary additions by Microsoft are to be deprecated (their support will be removed) in favor of open standard mechanisms for the embedding of objects, such as video and executable content. Netscape can support the embedding of video clips through its plugin mechanism using the `<EMBED>` element. See `<EMBED>` for more details.

CONTROLS

This attribute has no values. It is a flag that, if set, displays the standard Windows AVI Control Panel to allow the user to control the display of the video clip.

DYNSRC

This attribute specifies the address of a video clip to be displayed in the window. It stands for *dynamic source*.

```
<IMG SRC="filmclip.gif" DYNSRC="filmclip.avi">
```

Internet Explorer displays the movie `filmclip.avi`; other browsers display the image `filmclip.gif`.

The attributes used to control the playing of the video clip follow.

- **LOOP** Specifies how many times a video clip will loop when activated. If `n=-1` or if `LOOP=INFINITE` is specified, the video will loop indefinitely.
- **LOOPDELAY** Specifies in milliseconds how long a video clip will wait between play loops.

Note

Because the `DYNSRC` is an attribute of the `IMG` element, other attributes of the `IMG` element, such as `HEIGHT`, `WIDTH`, `HSPACE`, `VSPACE`, `BORDER`, and so on, also are acceptable, and if specified, will format the display window for the video clip.

- **START** Specifies when the video clip should start playing. It accepts values of `FILEOPEN` or `MOUSEOVER`. `FILEOPEN` means that the video will start playing as soon as it finishes downloading from the Web server or distribution source. This is the default value. `MOUSEOVER` means start playing when the user moves the mouse cursor over the animation. It is possible to specify both of these values together.

Inline VRML Worlds

Note

As with other `` related object embedding mechanisms (inline video), future versions of the Internet Explorer will support open standard object embedding mechanisms instead of relying on proprietary extensions.

Microsoft's Internet Explorer (from version 2) has added the capability to include inline embedded VRML viewable by installing the Virtual Explorer plugin module, available from the Microsoft Windows 95 Web site (<http://www.microsoft.com/windows>). It does this by adding

the VRML attribute to the element.

As the attribute is used in the element, it supports many of the other attributes of the element, such as HEIGHT, WIDTH, VSPACE, HSPACE, and so on.

For example;

```
<IMG SRC="picture.gif" VRML="world.wrl" HEIGHT=250  
WIDTH=300>
```

embeds the VRML world, `world.wrl`, into the HTML document, with the navigation controls below the embedding pane. The pane is displayed according to the dimensions specified. For browsers other than the Virtual Explorer (Internet Explorer with the VRML add-on), the picture `picture.gif` is displayed.

Note

Embedding of VRML worlds also is supported by Netscape, using the Netscape Live3D plugin module and the <EMBED> element. See <EMBED> for more details.

Information-Type and Character-Formatting Elements

The following information-type and character-formatting elements are supported by most browsers.

Note

Different information type elements may be rendered in the same way. The following are what are sometimes called *Logical formatting elements*. They suggest to the browser that the enclosed text should be rendered in a way set by the browser rather than physically fixing the display type. Elements that do this are character-formatting elements that produce strict rendering of the text.

Information-type elements:

<CITE>...</CITE>	Citation
<CODE>...</CODE>	An example of code
...	Emphasis
<KBD>...</KBD>	User typed text
<SAMP>...</SAMP>	A sequence of literal characters
...	Strong typographic emphasis
<VAR>...</VAR>	Indicates a variable name
<!-- ... -->	Defining comments

Character-formatting elements:

...	Boldface type
<BIG>...</BIG>	Big text
<BLINK>...</BLINK>	Blinking text
<I>...</I>	Italics
<SMALL>...</SMALL>	Small text
<STRIKE>...</STRIKE>	
(or <S>...</S>)	Strikethrough text
_{...}	Subscript
^{...}	Superscript

<code><TT>...</TT></code>	TypeType (or Teletype)
<code><U>...</U></code>	Underlined text

Although character-formatting elements (physical elements) may be nested within the content of other character-formatting elements, browsers are not required to render nested character-level elements distinctly from non-nested elements. For example,

```
plain <B>bold <I>italic</I></B>
```

may be rendered the same as

```
plain <B>bold </B><I>italic</I>
```

<!-- Comments -->

To include comments in an HTML document that will be ignored by the browser, surround them with `<!--` and `-->`. After the comment delimiter, all text up to the next occurrence of `-->` is ignored. Comments therefore cannot be nested. White space is allowed between the closing `--` and `>`, but not between the opening `<!--` and `--`. Comments can be used anywhere within an HTML document and generally are used as markers to improve the readability of complex HTML documents.

For example:

```
<HEAD>
<TITLE>The HTML Reference</TITLE>
<!-- Created by Stephen Le Hunte, April 1996 -->
</HEAD>
```

Note
Some browsers incorrectly consider a <code>></code> sign to terminate a comment.

`...`

The bold element specifies that the text should be rendered in boldface, where available. Otherwise, alternative mapping is allowed.

```
The instructions <B>must be read</B> before continuing.
```

renders as

The instructions **must be read** before continuing.

`<BIG>...</BIG>`

The `<BIG>` element specifies that the enclosed text should be displayed, if practical, using a big font (compared with the current font). This is an HTML 3.0 element and may not be widely supported. For example,

```
This is normal text, with <BIG>this bit</BIG> being big
text.
```

is rendered as

This is normal text, with **this bit** being big text.

Note
Use of this element currently is supported by Netscape and the Internet Explorer only. It also allows the <code><BIG>...</BIG></code> element to be used surrounding the <code><SUB>...</SUB></code> and <code><SUP>...</SUP></code> elements to force rendering of the sub/superscript text as normal size text as

opposed to the default, slightly smaller text normally used.

The exact appearance of the big text changes depending on any `` and `<BASEFONT SIZE= . . . >` settings, if specified.

<BLINK>

Surrounding any text with this element causes the selected text to blink on the viewing page. This can add extra emphasis to selected text.

`<BLINK>This text would blink on the page</BLINK>`

Note

The `<BLINK> . . . </BLINK>` element currently is supported only by Netscape.

<CITE> . . . </CITE>

The citation element specifies a citation and typically is rendered in an italic font. For example,

`This sentence contains a <CITE>citation reference</CITE>`

would look like this:

This sentence contains a *citation reference*

<CODE> . . . </CODE>

The code element should be used to indicate an example of code and typically is rendered in a monospaced font. This should not be confused with the preformatted text (`<PRE>`) element. For example,

The formula is: `<CODE>x=(-b+/- (b^2-4ac) ^1/2) /2a</CODE>`

would look like this:

The formula is: `x=(-b+/- (b^2-4ac)^1/2)/2a`

** . . . **

The emphasis element indicates typographic emphasis and typically is rendered in an italic font. For example,

The `Emphasis` element typically renders as Italics.

renders as

The *Emphasis* element typically renders as Italics.

<I> . . . </I>

The italic element specifies that the text should be rendered in italic font, where available. Otherwise, alternative mapping is allowed. For example,

Anything between the `<I>I elements</I>` should be italics.

renders as

Anything between the *I elements* should be italics.

<KBD> . . . </KBD>

The keyboard element can be used to indicate text to be typed by a user and typically is rendered in a monospaced font. It might commonly be used in an instruction manual. For example,

To login to the system, enter `<KBD>"GUEST"</KBD>` at the command prompt.

renders as

To login to the system, enter "GUEST" at the command prompt.

<SAMP> . . . </SAMP>

The sample element can be used to indicate a sequence of literal characters and typically is rendered in a monospaced font. For example,

A sequence of `<SAMP>literal characters</SAMP>` commonly renders in a monospaced

font.

renders as

A sequence of `literal characters` commonly renders in a mono spaced font.

<SMALL> . . . </SMALL>

The `<SMALL>` element specifies that the enclosed text should be displayed, if practical, using a small font (compared with the current font). This is an HTML 3.2 element and may not be widely supported. For example,

This is normal text, with `<SMALL>this bit</SMALL>` being small text.

rendereds as

This is normal text, with this bit being small text.

Note

Use of this element currently is supported by Netscape and the Internet Explorer only. They also allow the `<SMALL> . . . </SMALL>` element to be used surrounding the `_{. . .}` and `^{. . .}` elements to force rendering of the sub/superscript text as text even smaller than the default, slightly smaller (compared to the normal) text normally used.

The exact appearance of the small text changes depending on any `` and `<BASEFONT SIZE=. . . >` settings, if specified.

<STRIKE> . . . </STRIKE>

The `<STRIKE> . . . </STRIKE>` element states that the enclosed text should be displayed with a horizontal line striking through the text. Alternative mappings are allowed if this is not practical. This is an HTML 3.2 element and may not be widely supported. For example,

This text would be `<STRIKE>struck through</STRIKE>`

renders as

This text would be ~~struck through~~

Note

Although use of the `<STRIKE>` element currently is supported by Netscape and Mosaic, the element contained in current versions of the HTML 3.2 specification is `<S> . . . </S>`, which is supported by Mosaic but not Netscape. The Microsoft Internet Explorer supports either version of the element.

` . . . `

The strong element can be used to indicate strong typographic emphasis and typically is rendered in a bold font. For example,

The instructions `must be read` before continuing.

renders as

The instructions **must be read** before continuing.

`_{. . .}`

The `<SUB>` element specifies that the enclosed text should be displayed as a subscript and, if practical, using a smaller font (compared with normal text). This is an HTML 3.2 element and may not be widely supported. For example,

This is the main text, with `_{this bit}` being subscript.

renders as

This is the main text, with this bit being subscript.

Note

The selected text will be made a superscript to the main text, formatting the selected text slightly smaller than the normal text. Netscape and the Internet Explorer can be forced to make subscripts even smaller by compounding the `_{. . .}` element with the `<SMALL> . . . </SMALL>` element or to render the subscript the same size as the normal text by compounding the `_{. . .}` element with the `<BIG> . . . </BIG>` element.

The exact appearance of the subscript text changes depending on any `` and `<BASEFONT SIZE=. . .>` settings, if specified.

`^{. . .}`

The `<SUP>` element specifies that the enclosed text should be displayed as a superscript and, if practical, using a smaller font (compared with normal text). This is an HTML 3.2 element and may not be widely supported. For example,

This is the main text, with `^{this bit}` being superscript.

renders as

This is the main text, with this bit being superscript.

Note

The selected text will be made a superscript to the main text, formatting the selected text slightly smaller than the normal text. Netscape and the Internet Explorer can be forced to make superscripts even smaller by

compounding the `^{...}` element with the `<SMALL>...</SMALL>` element or to render the superscript the same size as the normal text by compounding the `^{...}` element with the `<BIG>...</BIG>` element.

The exact appearance of the superscript text changes depending on any `` and `<BASEFONT SIZE=...>` settings, if specified.

`<TT>...</TT>`

The Teletype element specifies that the text be rendered in fixed-width typewriter font where available. Otherwise, alternative mapping is allowed. For example,

Text between the `<TT>` TypeType elements`</TT>` should be rendered in fixed width typewriter font.

renders as

Text between the TypeType elements should be rendered in fixed-width typewriter font.

`<U>...</U>`

The `<U>` element states that the enclosed text should be rendered, if practical, underlined. This is an HTML 3.2 element and may not be widely supported. For example,

The `<U>`main point`</U>` of the exercise...

renders as

The main point of the exercise...

Note

Currently, Netscape doesn't support use of the `<U>` element.

`<VAR>...</VAR>`

The variable element can be used to indicate a variable name and typically is rendered in an italic font. For example,

When coding, `<VAR>`LeftIndent()`</VAR>` must be a variable

renders as

When coding, *LeftIndent()* must be a variable.

List Elements

HTML supports several types of lists, all of which may be nested. If used, they should be present in the `<BODY>` of an HTML document.

<code><DIR>...</DIR></code>	Directory list
<code><DL>...</DL></code>	Definition list
<code><MENU>...</MENU></code>	Menu list
<code>...</code>	Ordered list
<code>...</code>	Unordered list

`<DIR>...</DIR>`

A directory list element can be used to present a list of items, which may be arranged in columns, typically 24 characters wide. Some browsers attempt to optimize the column width as a

function of the widths of individual elements.

A directory list must begin with the `<DIR>` element, which is followed immediately by a `` (list item) element:

```
<DIR>
<LI>A-H
<LI>I-M
<LI>M-R
<LI>S-Z
</DIR>
```

`<DL> . . . </DL>`

Definition lists typically are rendered by browsers, with the definition term `<DT>` flush left in the display window with the definition data `<DD>` rendered in a separate paragraph, indented after the definition term. Individual browsers also may render the definition data on a new line, below the definition term.

Example of use:

```
<DL>
<DT>&lt;PRE&gt;<DD>Allows for the presentation of
preformatted text.
<DT>&lt;P&gt;<DD>This is used to define paragraph blocks.
</DL>
```

The layout of the definition list is at the discretion of individual browsers. However, generally, the `<DT>` column is allowed one-third of the display area. If the term contained in the `<DT>` definition exceeds this in length, it may be extended across the page with the `<DD>` section moved to the next line, or it may be wrapped onto successive lines of the left-hand column.

Single occurrences of a `<DT>` element without a subsequent `<DD>` element are allowed and have the same significance as if the `<DD>` element had been present with no text.

The opening list element must be `<DL>` and must be followed immediately by the first term (`<DT>`).

The definition list type can take the `COMPACT` attribute, which suggests that a compact rendering be used, implying that the list data may be large in order to minimize inefficient display window space. Generally, this is displayed table-like, with the definition terms and data rendered on the same line.

```
<DL COMPACT>
<DT>&lt;PRE&gt;<DD>Allows for the presentation of
preformatted text.
<DT>&lt;P&gt;<DD>This is used to define paragraph blocks.
</DL>
```

`<MENU> . . . </MENU>`

Menu lists typically are rendered as discrete items on a single line. It is more compact than the rendering of an unordered list. Typically, a menu list is rendered as a bulleted list, but this is at the discretion of the browser.

A menu list must begin with a `<MENU>` element, which is followed immediately by a `` (list item) element:

```
<MENU>
<LI>First item in the list.
<LI>Second item in the list.
<LI>Third item in the list.
```

</MENU>

 . . .

The ordered list element is used to present a numbered list of items, sorted by sequence or order of importance, and typically is rendered as a numbered list, but this is at the discretion of individual browsers.

Note

The list elements are not sorted by the browser when displaying the list. (This sorting should be done manually when adding the HTML elements to the desired list text.) Ordered lists can be nested.

An ordered list must begin with the element, which is followed immediately by a (list item) element:

```
<OL>
<LI>Click on the desired file to download.
<LI>In the presented dialog box, enter a name to save the
file with.
<LI>Click 'OK' to download the file to your local drive.
</OL>
```

The ordered list element can take the COMPACT attribute, which suggests that a compact rendering be used.

The average ordered list counts 1, 2, 3, ... and so on. The TYPE attribute allows authors to specify whether the list items should be marked with the following:

(TYPE=A)	Capital letters. For example, A, B, C ...
(TYPE=a)	Small letters. For example, a, b, c ...
(TYPE=I)	Large roman numerals. For example, I, II, III ...
(TYPE=i)	Small roman numerals. For example, i, ii, iii ...
(TYPE=1)	The default numbers. For example, 1, 2, 3 ...

For lists that you want to start at values other than 1, the new attribute START is available.

START always is specified in the default numbers and is converted based on TYPE before display. Thus, START=5 displays an E, e, V, v, or 5, based on the TYPE attribute. For example, changing the preceding example to

```
<OL TYPE=a START=3>
<LI>Click on the desired file to download.
<LI>In the presented dialog box, enter a name to save the
file with.
<LI>Click 'OK' to download the file to your local drive.
</OL>
```

presents the list as using lowercase letters, starting at c.

To give even more flexibility to lists, the TYPE attribute can be used with the element. It takes the same values as , and it changes the list type for that item and all subsequent items. For ordered lists, the VALUE attribute is also allowed, which can be used to set the count for that list item and all subsequent items.

Note

The TYPE attribute used in the element and the element and the START attribute in the element are supported only by Netscape and Internet Explorer.

 . . .

The unordered list element is used to present a list of items that typically are separated by white space and/or marked by bullets, but this is as the discretion of individual browsers.

An unordered list must begin with the element, which is followed immediately by a (list item) element. Unordered lists can be nested.

```
<UL>
<LI>First list item
<LI>Second list item
<LI>Third list item
</UL>
```

The unordered list element can take the COMPACT attribute, which suggests that a compact rendering be used.

The basic bulleted list has a default progression of bullet types that change as you move through indented levels: from a solid disc to a circle to a square. The TYPE attribute can be used in the element so that no matter what the indent level, the bullet type can be specified as the following:

```
TYPE=disc
TYPE=circle
TYPE=square
```

To give even more flexibility to lists, the TYPE attribute to the element also is allowed. It takes the same values as and it changes the list type for that item and all subsequent items.

Note

The TYPE attribute, when used in the and elements, is supported by Netscape only.

Tables

At present, the table HTML elements are

<TABLE> . . . </TABLE>	The table delimiter
<TR . . .> . . . </TR>	Specifies number of rows in a table
<TD . . .> . . . </TD>	Specifi table data cells
<TH . . .> . . . </TH>	Table header cell
<CAPTION . . .> . . . </CAPTION>	Specifies the table caption

Internet Explorer has introduced support for various HTML 3.0 table elements. Those introduced are

<THEAD> . . . </THEAD>	Specifies the table head
<TBODY> . . . </TBODY>	Specifies the table body
<TFOOT> . . . </TFOOT>	Specifies the table footer
<COLGROUP> . . . </COLGROUP>	Groups column alignments
<COL> . . . </COL>	Specifies individual column alignments

Also, some new attributes have been introduced. These are

<TABLE BACKGROUND=" . . . ">	Specifies a background image for the table
<TH BACKGROUND=" . . . ">	Specifies a background image for the table header
<TD BACKGROUND=" . . . ">	Specifies a background image for a table data cell
<TABLE FRAME=" . . . ">	Specifies the appearance of the table frame
<TABLE RULES=" . . . ">	Specifies the appearance of the table dividing lines

<TABLE> . . . </TABLE>

This is the main wrapper for all the other table elements; other table elements are ignored if they aren't wrapped inside a <TABLE> . . . </TABLE> element. By default, tables have no borders; borders are added if the BORDER attribute is specified.

The <TABLE> element has the following attributes.

ALIGN="left | right"

Some browsers (Internet Explorer and Netscape) support the ALIGN attribute to the <TABLE> element. Like that used for *floating images*, it allows a table to be aligned to the left or right of the page, allowing text to flow around the table. Also, as with floating images, it is necessary to have knowledge of the <BR CLEAR= . . . > element to be able to organize the text display to minimize poor formatting.

BACKGROUND

Internet Explorer supports the placement of images in the <TABLE> element (also in the <TD> and <TH> elements). If used in the <TABLE> element, the image in question is tiled behind all the table cells. Any of the supported graphics file formats can be used as a graphic behind a table.

BGCOLOR="#rrggb| color name"

Internet Explorer and Netscape support use of this attribute (also supported in the <BODY> element). It allows the background color of the table to be specified, using the specified *color names* or an *rrggb* hexadecimal triplet.

BORDER

This attribute can be used to control and set the borders to be displayed for the table. If present, a border is drawn around all data cells. The exact thickness and display of this default border is at the discretion of individual browsers. If the attribute isn't present, the border is not displayed, but the table is rendered in the same position as if there were a border (allowing room for the border). It also can be given a value, BORDER=<value>, which specifies the thickness of the table border. The border value can be set to 0, which regains all the space that the browser has set aside for any borders (as in the case where no border has been set).

BORDERCOLOR="#rrggb| color name"

Internet Explorer includes support for this attribute, which sets the border color of the table. Any of the predefined color names can be used, as well as any color defined by an *rrggb* hexadecimal triplet. It is necessary for the BORDER attribute to be present in the main <TABLE> element for border coloring to work.

BORDERCOLORDARK="#rrggb| color name"

Internet Explorer enables you to use the BORDERCOLORDARK attribute to set the darker color to be displayed on a 3-D <TABLE> border. It is the opposite of BORDERCOLORLIGHT. Any of the predefined color names can be used, as well as any color defined by an *rrggb* hexadecimal triplet. It is necessary for the BORDER attribute to be present in the main <TABLE> element for border coloring to work.

Note

The BGCOLOR, BORDERCOLOR, BORDERCOLORLIGHT, and BORDERCOLORDARK attributes also can be used in <TH>, <TR>, and <TD> elements, with the color defined in the last element over-riding those defined before. For example, if a <TD> element contains a BORDERCOLOR attribute setting, the setting specified will be used instead

of any color settings that may have been specified in the <TR> element, which over-rides any color settings in the <TABLE> element.

BORDERCOLORLIGHT="#rrggbb|color name"

Internet Explorer enables you to use the BORDERCOLORLIGHT attribute to set, the lighter color to be displayed on a 3-D <TABLE> border. It is the opposite of BORDERCOLORDARK. Any of the predefined color names can be used, as well as any color defined by an rrggbb hexadecimal triplet. It is necessary for the BORDER attribute to be present in the main <TABLE> element for border coloring to work.

CELLPADDING=value

The CELLPADDING is the amount of white space between the borders of the table cell and the actual cell data (whatever is to be displayed in the cell). It defaults to an effective value of 1. This example gives the most compact table possible:

```
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0>
```

CELLSPACING=value

The CELLSPACING is the amount of space inserted between individual table data cells. It defaults to an effective value of 2.

FRAME

Only Internet Explorer supports the use of this attribute. It requires the BORDER attribute to be set and affects the display of the table borders. It can accept any of the following values:

void	Removes all the external borders
above	Displays external borders at the top of the table only
below	Displays external borders at the bottom of the table only
hsides	Displays external borders at the horizontal sides of the table-at the top and bottom of the table
lhs	Displays external borders at the left-hand edge of the table only
rhs	Displays external borders at the right-hand edge of the table only
vsides	Displays external borders at both left-and right-hand edges of the table
box	Displays a box around the table (top, bottom, left-and right-hand sides)

HEIGHT=value_or_percent

If used, this attribute can specify the exact height of the table in pixels or the height of the table as a percentage of the browser display window.

RULES

Internet Explorer supports this new attribute. It requires the BORDER value to be set and may only be used in tables where the <THEAD>, <TBODY>, and <TFOOT> sections have been set. It affects the display of the internal table borders (rules). It can accept the following values:

none	Removes all the internal rules
basic	Displays horizontal borders between the <THEAD>, <TBODY>, and <TFOOT> sections
rows	Displays horizontal borders between all rows
cols	Displays horizontal borders between all columns
all	Displays all the internal rules

VALIGN="top|bottom"

The Internet Explorer supports this attribute that specifies the vertical alignment of the text displayed in the table cells. The default, which is used if the attribute is not set, is center aligned.

WIDTH=*value_or_percent*

If used, this attribute can specify the exact width of the table in pixels or the width of the table as a percentage of the browser display window.

<CAPTION ...>...</CAPTION>

This represents the caption for a table. <CAPTION> elements should appear inside the <TABLE> but not inside table rows or cells. The caption accepts an alignment attribute that defaults to `ALIGN=top` but can be explicitly set to `ALIGN=bottom`. Like table cells, any document body HTML can appear in a caption. Captions by default are horizontally centered with respect to the table, and they may have their lines broken to fit within the width of the table.

The <CAPTION> element can accept the following attributes.

ALIGN="*top | bottom | left | center | right***"**

The `ALIGN` attribute controls whether the caption appears above or below the table, using the `top` and `bottom` values, defaulting to `top`. The Internet Explorer allows the <CAPTION> element to be left, right, or center aligned. For the Internet Explorer to set the <CAPTION> at the top or bottom of the table, it is necessary to use the `VALIGN` attribute.

VALIGN="*top | bottom***"**

The Internet Explorer allows use of the `VALIGN` attribute inside the <CAPTION> element. It specifies whether the caption text should be displayed at the top or bottom of the table.

<COL> ... </COL>

This element, which is Internet Explorer specific, can be used to specify the text alignment for table columns. It accepts the following attributes.

- **ALIGN="***center | justify | left | right***"** Sets the text alignment within the column group. The default value is `center`.
- **SPAN=***value* Sets the number of columns upon which the `ALIGN` attribute is to act.

<COLGROUP> ... </COLGROUP>

This element, which is Internet Explorer specific, can be used to group columns to set their alignment properties. It accepts the following attributes:

- **ALIGN="***center | justify | left | right***"** Sets the text alignment within the column group. The default value is `center`.
- **SPAN=***value* Sets the number of columns upon which the `ALIGN` and `VALIGN` attributes are to act.
- **VALIGN="***baseline | bottom | middle | top***"** Sets the vertical text alignment within the column group.

<TBODY> ... </TBODY>

This element, which is Internet Explorer specific, is used to specify the body section of the table. It is somewhat analogous to the <BODY> element. It directly affects the rendering of the table on-screen, but is required if you want `RULES` to be set in the <TABLE>.

<TD ...>...</TD>

This stands for *table data*, and specifies a standard table data cell. Table data cells must only appear within table rows. Each row need not have the same number of cells specified because, short rows will be padded with blank cells on the right. A cell can contain any of the HTML

elements normally present in the body of an HTML document.

Internet Explorer allows the use of `<TD></TD>` to specify a blank cell, that will be rendered with a border (if a border has been set). Other browsers require some character within a data cell for it to be rendered with a border.

`<TD ...>...</TD>` can accept the following attributes:

- **ALIGN="left | center | right"** Controls whether text inside the table cell(s) is aligned to the left, right, or centered within the cell.
- **BACKGROUND** Internet Explorer supports the placing of images inside the `<TD>` element (also in the `<TABLE>`, `<TD>` and `<TH>` elements). If used in the `<TD>` element, the image in question is tiled behind the particular data cell. Any of the supported graphics file formats can be used as a graphic behind a table.
- **BGCOLOR="#rrggbb | color name"** Internet Explorer and Netscape support the use of this attribute (also supported in the `<BODY>` element). It allows the background color of the data cell to be specified, using the specified color names, or an `rrggbb` hexadecimal triplet.
- **BORDERCOLOR="#rrggbb | color name"** Internet Explorer includes support for this attribute, which sets the border color of the data cell. Any of the predefined color names can be used, as well as any color defined by an `rrggbb` hexadecimal triplet. It is necessary for the **BORDER** attribute to be present in the main `<TABLE>` element for border coloring to work.
- **BORDERCOLORDARK="#rrggbb | color name"** Internet Explorer allows the **BORDERCOLORDARK** attribute independently, so that the darker color is displayed on a 3-D `<TD>` border. It is the opposite of **BORDERCOLORLIGHT**. Any of the predefined color names can be used, as well as any color defined by an `rrggbb` hexadecimal triplet. It is necessary for the **BORDER** attribute to be present in the main `<TABLE>` element for border coloring to work.

Note

The **BGCOLOR**, **BORDERCOLOR**, **BORDERCOLORDARK**, and **BORDERCOLORLIGHT** attributes also can be used in `<TABLE>`, `<TH>`, and `<TR>` elements, with the color defined in the last element over-riding those defined before. For example, if a `<TD>` element contains a **BORDERCOLOR** attribute setting, the setting specified is used instead of any color settings that may have been specified in the `<TR>` element, which in turn overrides any color settings in the `<TABLE>` element.

- **BORDERCOLORLIGHT="#rrggbb | color name"** Internet Explorer allows you to use the **BORDERCOLORLIGHT** attribute to set, the lighter color to be displayed on a 3-D `<TD>` border. It is the opposite of **BORDERCOLORDARK**. Any of the predefined color names can be used, as well as any color defined by an `rrggbb` hexadecimal triplet. It is necessary for the **BORDER** attribute to be present in the main `<TABLE>` element for border coloring to work.
- **COLSPAN="value"** This attribute can appear in any table cell (`<TH>` or `<TD>`), and it specifies how many columns of the table this cell should span. The default **COLSPAN** for any cell is 1.
- **HEIGHT=value_or_percent** If used, this attribute can specify the exact height of the data cell in pixels or the height of the data cell as a percentage of the browser display window. Only one data cell can set the height for an entire row, typically being the last data cell to be rendered.

- **NOWRAP** If this attribute appears in any table cell (<TH> or <TD>), it means the lines within this cell cannot be broken to fit the width of the cell. Be cautious when using this attribute, because it can result in excessively wide cells.
- **ROWSPAN="value"** This attribute can appear in any table cell (<TH> or <TD>), and it specifies how many rows of the table this cell should span. The default ROWSPAN for any cell is 1. A span that extends into rows that were never specified with a <TR> is truncated.
- **VALIGN="top|middle|bottom|baseline"** The VALIGN attribute controls whether text inside the table cell(s) is aligned to the top, bottom, or vertically centered within the cell. It also can specify that all the cells in the row should be vertically aligned to the same baseline.
- **WIDTH=value_or_percent** If used, this attribute can specify the exact width of the data cell in pixels or the width of the data cell as a percentage of the table being displayed. Only one data cell can set the width for an entire column, typically being the last data cell to be rendered.

<TFOOT>...</TFOOT>

This element, which is Internet Explorer specific, is used to specify the footer section of the table. It directly affects the rendering of the table on-screen, but is required if you want RULES to be set in the <TABLE> .

<TH ...>...</TH>

This stands for *table header*. Header cells are identical to data cells in all respects, with the exception that header cells are in a bold font and have a default ALIGN=center.

<TH ...>...</TH> can contain the following attributes:

- **ALIGN="left|center|right"** Controls whether text inside the table cell(s) is aligned to the left, right, or centered within the cell.
- **BACKGROUND** Internet Explorer supports the placing of images inside the <TH> element (also in the <TABLE>, <TD>, and <TH> elements). If used in the <TH> element, the image in question is tiled behind the particular data cell. Any of the supported graphics file formats can be used as a graphic behind a table.
- **BGCOLOR="#rrggbb|color name"** Internet Explorer and Netscape support use of this attribute (also supported in the <BODY> element). It allows the background color of the header cell to be specified, using the specified color names, or an rrggbb hexadecimal triplet.
- **BORDERCOLOR="#rrggbb|color name"** Internet Explorer includes support for this attribute, which sets the border color of the header cell. Any of the predefined color names can be used, as well as any color defined by an rrggbb hexadecimal triplet. It is necessary for the BORDER attribute to be present in the main <TABLE> element for border coloring to work.
- **BORDERCOLORDARK="#rrggbb|color name"** Internet Explorer allows you to use the BORDERCOLORDARK attribute to set the darker color to be displayed on a 3-D <TH> border. It is the opposite of BORDERCOLORLIGHT. Any of the predefined color names can be used, as well as any color defined by an rrggbb hexadecimal triplet. It is necessary for the BORDER attribute to be present in the main <TABLE> element for border coloring to work.

Note

The **BGColor**, **bordercolor**, **bordercolordark**, and **bordercolorlight** attributes also can be used in **<table>**, **<td>**, and **<tr>** elements, with the color defined in the last element over-riding those defined before. For example, if a **<td>** element contains a **bordercolor** attribute setting, the setting specified is used instead of any color settings that may have been specified in the **<tr>** element, which overrides any color settings in the **<table>** element.

- **BORDERCOLORLIGHT="#rrggbb|color name"** Internet Explorer allows you to use the **BORDERCOLORLIGHT** attribute to set the lighter color to be displayed on a 3-D **<th>** border. It is the opposite of **BORDERCOLORDARK**. Any of the predefined color names can be used, as well as any color defined by an **rrggbb** hexadecimal triplet. It is necessary for the **BORDER** attribute to be present in the main **<table>** element for border coloring to work.
- **COLSPAN="value"** This attribute can appear in any table cell (**<th>** or **<td>**), and it specifies how many columns of the table this cell should span. The default **COLSPAN** for any cell is 1.
- **HEIGHT=value_or_percent** If used, this attribute can specify the exact height of the data cell in pixels or the height of the data cell as a percentage of the browser display window. Only one data cell can set the height for an entire row, typically being the last data cell to be rendered.
- **NOWRAP** Specifies that the lines within this cell cannot be broken to fit the width of the cell. Be cautious when using this attribute because it can result in excessively wide cells.
- **ROWSPAN="value"** This attribute can appear in any table cell (**<th>** or **<td>**), and it specifies how many rows of the table this cell should span. The default **ROWSPAN** for any cell is 1. A span that extends into rows that were never specified with a **<tr>** are truncated.
- **VALIGN="top|middle|bottom|baseline"** The **VALIGN** attribute controls whether text inside the table cell(s) is aligned to the top, bottom, or vertically centered within the cell. It also can specify that all the cells in the row should be vertically aligned to the same baseline.
- **WIDTH=value_or_percent** If used, this attribute can specify the exact width of the data cell in pixels or the width of the data cell as a percentage of the table being displayed. Only one data cell can set the width for an entire column, typically being the last data cell to be rendered.

<thead> . . . </thead>

This element, which is Internet Explorer specific, is used to specify the head section of the table. It is somewhat similar to the **<head>** element. It does directly affect the rendering of the table on-screen, but is required if you want **RULES** to be set in the **<table>**.

<tr . . .> . . . </tr>

This stands for *table row*. The number of rows in a table is exactly specified by how many **<tr>** elements are contained within it, regardless of cells that may attempt to use the **ROWSPAN** attribute to span into non-specified rows.

The **<tr>** element can have the following attributes:

- **ALIGN="left|center|right"** Controls whether text inside the table cell(s) is aligned to the left, right, or center of the cell.

- **BGCOLOR="#rrggbb|color name"** Internet Explorer and Netscape support use of this attribute (also supported in the <BODY> element). It allows the background color of the table to be specified, using the specified color names, or an rrggbb hexadecimal triplet.
- **BORDERCOLOR="#rrggbb|color name"** Internet Explorer includes support for this attribute, which sets the border color of the row. Any of the predefined color names can be used, as well as any color defined by an rrggbb hexadecimal triplet. It is necessary for the BORDER attribute to be present in the main <TABLE> element for border coloring to work.
- **BORDERCOLORDARK="#rrggbb|color name"** Internet Explorer allows you to use the BORDERCOLORDARK attribute to set the darker color to be displayed on a 3-D <TR> border. It is the opposite of BORDERCOLORLIGHT. Any of the predefined color names can be used, as well as any color defined by an rrggbb hexadecimal triplet. It is necessary for the BORDER attribute to be present in the main <TABLE> element for border coloring to work.

Note

The BGCOLOR, BORDERCOLOR, BORDERCOLORLIGHT, and BORDERCOLORDARK attributes also can be used in <TABLE>, <TH>, and <TD> elements, with the color defined in the last element over-riding those defined before. If a <TD> element contains a BORDERCOLOR attribute setting, the setting specified is used instead of any color settings that may have been specified in the <TR> element, which overrides any color settings in the <TABLE> element.

- **BORDERCOLORLIGHT="#rrggbb|color name"** Internet Explorer allows you to use the BORDERCOLORLIGHT attribute to set the lighter color to be displayed on a 3-D <TR> border. It is the opposite of BORDERCOLORDARK. Any of the predefined color names can be used, as well as any color defined by an rrggbb hexadecimal triplet. It is necessary for the BORDER attribute to be present in the main <TABLE> element for border coloring to work.
- **VALIGN="top|middle|bottom|baseline"** Controls whether text inside the table cell(s) is aligned to the top, bottom, or vertically centered within the cell. It also can specify that all the cells in the row should be vertically aligned to the same baseline.

Table Examples

Here are some sample HTML <TABLE> fragments with accompanying screenshots.

A Simple Table

```
<TABLE BORDER>
<TR>
<TD>Data cell 1</TD><TD>Data cell 2</TD>
</TR>
<TR>
<TD>Data cell 3</TD><TD>Data cell 4</TD>
</TR>
</TABLE>
```

A Table Using ROWSPAN

```
<TABLE BORDER>
<TR>
<TD ROWSPAN=2>This cell spans two rows</TD>
<TD>These cells</TD><TD>would</TD>
</TR>
```

```
<TR>
<TD>contain</TD><TD>other data</TD>
</TR>
</TABLE>
```

A Table Using COLSPAN

```
<TABLE BORDER>
<TR>
<TD>Data cell 1</TD>
<TD COLSPAN=2>This cell spans 2 columns</TD>
</TR>
<TR>
<TD>Data cell 2</TD><TD>Data cell 3</TD><TD>Data cell
4</TD>
</TR>
</TABLE>
```

A Table Using Headers

```
<TABLE BORDER>
<TR>
<TH>Netscape</TH><TH>Internet Explorer</TH><TH>Mosaic</TH>
</TR>
<TR>
<TD>X</TD><TD>X</TD><TD>--</TD>
</TR>
<TR>
<TD>X</TD><TD>--</TD><TD>X</TD>
</TR>
</TABLE>
```

A Table Using All of the Above

```
<TABLE BORDER>
<TR>
<TD><TH ROWSPAN=2></TH>
<TH COLSPAN=3>Browser</TH></TD>
</TR>
<TR>
<TD><TH>Netscape</TH><TH>Internet
Explorer</TH><TH>Mosaic</TH></TD>
</TR>
<TR>
<TH ROWSPAN=2>Element</TH>
<TH>&lt;DFN&gt;</TH><TD>--</TD><TD>X</TD><TD>--</TD>
</TR>
<TR>
<TH>&lt;DIR&gt;</TH><TD>X</TD><TD>X</TD><TD>X</TD>
</TR>
</TABLE>
```

A Table Using ALIGN/VALIGN

This table adds ALIGN and VALIGN attributes to the preceding example to improve the layout of the table.

```
<TABLE BORDER>
<TR>
<TD><TH ROWSPAN=2></TH>
<TH COLSPAN=3>Browser</TH></TD>
```

```
</TR>
<TR>
<TD><TH>Netscape</TH><TH>Internet
Explorer</TH><TH>Mosaic</TH></TD>
</TR>
<TR>
<TH ROWSPAN=2 VALIGN=top>Element</TH>
<TH>&lt;DFN&gt;</TH>
<TD ALIGN=center>-</TD>
<TD ALIGN=center>X</TD>
<TD ALIGN=center>-</TD>
</TR>
<TR>
<TH>&lt;DIR&gt;</TH>
<TD ALIGN=center>X</TD>
<TD ALIGN=center>X</TD>
<TD ALIGN=center>X</TD>
</TR>
</TABLE>
```

Nested Tables

To show that tables can be nested within each other. This table uses the ROWSPAN table, including the simple table inside one of the data cells.

```
<TABLE BORDER>
<TR>
<TD ROWSPAN=2>This cell spans two rows
<TABLE BORDER>
<TR>
<TD>Data cell 1</TD><TD>Data cell 2</TD>
</TR>
<TR>
<TD>Data cell 3</TD><TD>Data cell 4</TD>
</TR>
</TABLE>
</TD>
<TD>These cells</TD><TD>would</TD>
</TR>
<TR>
<TD>contain</TD><TD>other data</TD>
</TR>
</TABLE>
```

Floating Tables

```
<TABLE ALIGN=left BORDER WIDTH=50%>
<TR>
<TD>This is a two row table</TD>
</TR>
<TR>
<TD>It is aligned to the left of the page</TD>
</TR>
</TABLE>
This text will be to the right of the table, and will fall
neatly beside the table
<BR CLEAR=all>
<HR>
<TABLE ALIGN=right BORDER WIDTH=50%>
<TR>
<TD>This is a two row table</TD>
```

```
</TR>
<TR>
<TD>It is aligned to the right of the page</TD>
</TR>
</TABLE>
This text will be to the left of the table, and will fall
neatly beside the table
<BR CLEAR=all>
<HR>
```

A Colored Table

```
<TABLE BORDER BGCOLOR=Silver BORDERCOLOR=Black WIDTH=50%>
<TR>
<TD>This is the first cell</TD>
<TD>This is the second cell</TD>
</TR>
<TR BORDERCOLOR=Red BGCOLOR=Green>
<TD>This is the third cell</TD>
<TD>This is the fourth cell</TD>
</TR>
<TR BORDERCOLOR=Red BGCOLOR=Green>
<TD BORDERCOLOR=Yellow>This is the fifth cell</TD>
<TD BGCOLOR=White>This is the sixth cell</TD>
</TR>
</TABLE>
```

An HTML 3.2 Table

Note
The following HTML table is at present only supported by Internet Explorer.

```
<TABLE BORDER FRAME=hsides RULES=cols>
<COL ALIGN=left>
<COLGROUP SPAN=3 ALIGN=center VALIGN=middle>
<THEAD>
<CAPTION ALIGN=center><FONT SIZE=+1><B>A section of the
Comparison Table</B>
</FONT>
</CAPTION>
<TR>
<TD>Element</TD><TD><B>Internet
Explorer</B></TD><TD><B>Netscape</B>
</TD><TD><B>Mosaic</B></TD>
</TR>
</THEAD>
<TBODY>
<TR>
<TD>&lt;B>&gt;</TD><TD>X</TD><TD>X</TD><TD>X</TD>
</TR>
<TR>
<TD>&lt;BASE ...&gt;</TD><TD>X</TD><TD>X</TD><TD>X</TD>
</TR>
<TR>
<TD>...HREF</TD><TD>X</TD><TD>X</TD><TD>X</TD>
</TR>
<TR>
<TD>...TARGET</TD><TD>X</TD><TD>X</TD><TD></TD>
</TR>
<TR>
```

```
<TD>&lt;BASEFONT ...&gt;</TD><TD>X</TD><TD>X</TD><TD></TD>
</TR>
<TR>
<TD VALIGN=top> ...SIZE</TD><TD>X<BR><FONT SIZE=-1>(only
visible<BR>when FONT<BR>SIZE= used<BR>as
well)</FONT></TD><TD VALIGN=top>X</TD><TD></TD>
</TR>
<TR>
<TD> ...FACE</TD><TD>X</TD><TD></TD><TD></TD>
</TR>
<TR>
<TD VALIGN=top>&lt;BGSOUND ...&gt;</TD><TD
VALIGN=top>X</TD><TD>
</TD><TD>X<BR><FONT SIZE=-1>(will spawn<BR>player
for<BR>.mid files)
</FONT></TD>
</TR>
</TBODY>
<TFOOT></TFOOT>
</TABLE>
```

Apéndice B: Cross-Browser Comparison of HTML

Table G.1 lists all the HTML 3.2 elements and attributes and indicates which browsers support them.

Table G.1. Comparison of HTML across three popular browsers.

<i>Element</i>	<i>Attribute</i>	<i>Internet Explorer</i>	<i>Netscape Navigator</i>	<i>NCSA Mosaic</i>
<!-- ...>		*	*	*
<!DOCTYPE ...>		*	*	*
<A ...>		*	*	*
	...HREF	*	*	*
	Mailto : ...TITLE			*
	...NAME	*	*	*
	...TITLE	*	*	*
	...REL	*	*	*
	...REV	*	*	*
	...URN	*	*	*
	...METHODS	*	*	*
	...TARGET		*	
<ADDRESS>		*	*	*
<APPLET ...>			*	
	...CODEBASE		*	
	...CODE		*	
	...ALT		*	
	...NAME		*	
	...WIDTH/HEIGHT		*	
	...ALIGN		*	
	...VSPACE/HSPACE		*	
	...PARAM NAME/VALUE		*	
		*	*	*
<BASE ...>		*	*	*
	...HREF	*	*	*
	...TARGET		*	
<BASEFONT ...>		*	*	
	...SIZE	*	*	
	(only visible			
		when FONT SIZE=used too)		
	...FACE	*		
<BG SOUND ...>		*		*
				(will spawn player for .mid files)
	...LOOP	*		*
	...DELAY			*
<BIG>		*	*	
<BLINK>			*	

<i>Element</i>	<i>Attribute</i>	<i>Internet Explorer</i>	<i>Netscape Navigator</i>	<i>NCSA Mosaic</i>
<BLOCKQUOTE>		*	*	*
<BODY ...>		*	*	*
	...BACKGROUND	*	*	*
	...TEXT	*	*	*
				(using color names is reliable)
	...LINK	*	*	*
	...VLINK	*	*	*
	...ALINK		*	
	...BGCOLOR	*	*	*
	...BGPROPERTIES	*		
	...LEFTMARGIN	*		
	...TOPMARGIN	*		
 		*	*	*
	...CLEAR	*	*	
<CAPTION>		*	*	*
	...ALIGN	*	*	*
		(top, bottom, left, right, center)	(top, bottom)	(top, bottom)
	...VALIGN	*		
		(top, bottom)		
<CENTER>		*	*	*
<CITE>		*	*	*
<CODE>		*	*	*
<COL>		*		
	...SPAN	*		
	...ALIGN	*		
<COLGROUP>		*		
	...SPAN	*		
	...ALIGN	*		
	...VALIGN	*		
<COMMENT>		*		*
<DFN>		*		
<DIR>		*	*	*
		(no bullet)		
<DIV>			*	
	...ALIGN		*	
			(left, right, center)	
<DL>		*	*	*
<DT>		*	*	*
<DD>		*	*	*
	...COMPACT		*	
<DT>		*	*	*
		*	*	*
<EMBED ...>			*	
		*	*	
	...SIZE	*	*	
	...COLOR	*	*	

<i>Element</i>	<i>Attribute</i>	<i>Internet Explorer</i>	<i>Netscape Navigator</i>	<i>NCSA Mosaic</i>
	...FACE	*		
<FORM>		*	*	*
<FRAME ...>			*	
	...SRC		*	
	...NAME		*	
	...MARGINWIDTH		*	
	...MARGINHEIGHT		*	
	...SCROLLING		*	
	...NORESIZE		*	
	...FRAMEBORDER	*		
	...FRAMESPACING	*		
<FRAMESET ...>		*	*	
	...ROWS	*	*	
	...COLS	*	*	
<H ALIGN= ...>		*	*	*
		(center only)	(right, left, center)	(right, left, center)
<H1>		*	*	*
<H2>		*	*	*
<H3>		*	*	*
<H4>		*	*	*
<H5>		*	*	*
<H6>		*	*	*
<HEAD>		*	*	*
<HR ...>		*	*	*
	...SIZE	*	*	*
	...WIDTH	*	*	*
	...ALIGN	*	*	*
	...NOSHADE	*	*	*
	...COLOR	*		
<HTML>		*	*	*
<I>		*	*	*
		*	*	*
	...ALIGN	*	*	*
				(top, middle, bottom only)
	...ALT	*	*	*
	...ISMAP	*	*	*
	...SRC	*	*	*
	...WIDTH	*	*	*
	...HEIGHT	*	*	*
				(image can't be distorted)
	...BORDER	*	*	
		(only when image is a link)		
	...VSPACE	*	*	
	...HSPACE	*	*	
	...LOWSRC		*	

<i>Element</i>	<i>Attribute</i>	<i>Internet Explorer</i>	<i>Netscape Navigator</i>	<i>NCSA Mosaic</i>
	...USEMAP	*	*	*
	...VRML	*		
<INPUT ...>		*	*	*
	...ALIGN	*	*	*
	...CHECKED	*	*	*
	...MAXLENGTH	*	*	*
	...NAME	*	*	*
	...SIZE	*	*	*
	...SRC	*	*	*
	...TYPE	*	*	*
	...VALUE	*	*	*
<ISINDEX ...>		*	*	*
	...PROMPT	*	*	
<KBD>		*	*	*
		*	*	*
<LINK ...>		*	*	*
<LISTING>		*	*	*
		(will translate special characters)	(renders 132 characters to the line and translates special characters)	
<MAP ...>		*	*	*
	...SHAPE	*	*	*
	...COORDS	*	*	*
	...AREA	*	*	*
<MARQUEE ...>		*		
	...ALIGN	*		
	...BEHAVIOR	*		
	...BGCOLOR	*		
	...DIRECTION	*		
	...HEIGHT	*		
	...WIDTH	*		
	...HSPACE	*		
	...LOOP	*		
	...SCROLLAMOUNT	*		
	...SCROLLDELAY	*		
	...VSPACE	*		
<MENU>		*	*	*
		(no bullet)		
<META ...>		*	*	*
	...HTTP-EQUIV	*	*	*
	...NAME	*	*	*
	...CONTENT	*	*	*
<NEXTID ...>		*	*	*
<NOBR>		*	*	
<NOFRAMES>		*	*	
<OBJECT>		*		
<PARAM>		*		

<i>Element</i>	<i>Attribute</i>	<i>Internet Explorer</i>	<i>Netscape Navigator</i>	<i>NCSA Mosaic</i>
<OL ...>		*	*	*
	...TYPE	*	*	
	...START	*	*	
	...VALUE	*	*	
<OPTION>		*	*	*
<P>		*	*	*
	...ALIGN	*	*	*
		(center only)	(left, right, center)	(left, right, center)
<PLAINTEXT>		*	*	*
		(allows closing element)		(allows closing element)
<PRE>		*	*	*
<S>		*		*
<SAMP>		*	*	*
<SCRIPT ...>		*	*	
	...LANGUAGE	*	*	
	...SRC	*	*	
<SELECT>		*	*	*
<SMALL>		*	*	
<SOUND ...>				*
				(.wav only)
	...SRC			*
	...DELAY			*
<STRIKE>		*	*	*
		*	*	*
<SUB>		*	*	*
<SUP>		*	*	*
<TABLE ...>		*	*	*
	...BORDER	*	*	*
	...CELLSPACING	*	*	*
	...CELLPADDING	*	*	*
	...WIDTH	*	*	*
	...HEIGHT	*	*	*
	...ALIGN	*	*	
	...VALIGN/FONT	*		
	...BGCOLOR	*		
	...BORDERCOLOR	*		
	...BORDERCOLORLIGHT	*		
	...BORDERCOLORDARK	*		
	...BACKGROUND	*		
	...FRAME	*		
	...RULES	*		
<TBODY>		*		
<TD ...>		*	*	*
	...ROWSPAN	*	*	*
	...COLSPAN	*	*	*

<i>Element</i>	<i>Attribute</i>	<i>Internet Explorer</i>	<i>Netscape Navigator</i>	<i>NCSA Mosaic</i>
	...ALIGN	*	*	
	...VALIGN	*	*	*
	...WIDTH	*	*	
	...HEIGHT	*		
	...NOWRAP	*	*	
	...BGCOLOR	*		
	...BORDERCOLOR	*		
	...BORDERCOLORLIGHT	*		
	...BORDERCOLORDARK	*		
	...BACKGROUND	*		
<TEXTAREA ...>		*	*	*
	...NAME	*	*	*
	...ROWS	*	*	*
	...COLS	*	*	*
	...WRAP		*	
<TFOOT>		*		
<TH ...>		*	*	*
	...ROWSPAN	*	*	*
	...COLSPAN	*	*	*
	...ALIGN	*	*	*
	...VALIGN	*	*	*
	...WIDTH	*	*	
	...HEIGHT	*		
	...NOWRAP	*	*	
	...BGCOLOR	*		
	...BORDERCOLOR	*		
	...BORDERCOLORLIGHT	*		
	...BORDERCOLORDARK	*		
	...BACKGROUND	*		
<THEAD>		*		
<TITLE>		*	*	*
<TR ...>		*	*	*
	...ALIGN	*	*	*
	...VALIGN	*	*	*
	...BGCOLOR	*		
	...BORDERCOLOR	*		
	...BORDERCOLORLIGHT	*		
	...BORDERCOLORDARK	*		
<TT>		*	*	*
<U>		*		*
		*	*	*
<VAR>		*	*	*
<WBR>		*	*	
<XMP>		*	*	*

<i>Element</i>	<i>Attribute</i>	<i>Internet Explorer</i>	<i>Netscape Navigator</i>	<i>NCSA Mosaic</i>
		(will translate special characters)		

Apéndice C: Internet Explorer-Specific HTML Extensions

CONTENTS

- [Adding a Background Sound](#)
- [Creating a Watermark](#)
- [Controlling Your Margins](#)
- [Adding Color](#)
 - [Table Backgrounds and Borders](#)
 - [Rule](#)
- [Table Sections and Column Properties](#)
 - [Table Sections](#)
 - [Setting Column Properties](#)
 - [Other Attributes of the <TABLE> Tag](#)
- [Inline AVI Support](#)
- [Scrolling Marquees](#)
- [Floating Frames](#)
- [HTML Layout Control](#)

Not to be outdone by Netscape, Microsoft has introduced its own browser-specific extensions to HTML. Like Netscape, Microsoft has submitted most of these extensions to the World Wide Web Consortium for consideration in future versions of the HTML standard. Don't be too surprised if you see some of the tags and attributes in this chapter showing up as standard HTML in later versions of this book.

In the meantime, if you're designing pages for visitors that are primarily using Internet Explorer 3, you can make use of the HTML extensions discussed in this chapter to enrich your pages and your audience's experience.

NOTE

While it has generated a number of extensions to HTML, Microsoft professes to recognize the importance of an open and uniform HTML standard. As part of its commitment to this, Microsoft has agreed to:

- Submit all HTML extensions to the World Wide Web Consortium before releasing them.
- Implement standard HTML as defined by the W3C.
- Identify any supported HTML tag that is not yet approved by the W3C as such.
- Publish an SGML Document Type Definition (DTD) for Internet

Explorer.

- Adhere to SGML architecture principles when developing and proposing new extensions.

Microsoft's explicit statement of this agreement demonstrates its willingness to be a "team player" in the advancement of HTML, while still being able to "push the envelope" when it comes to developing new HTML to distinguish its browser from others.

To read the full text of Microsoft's statement, direct your browser to

<http://www.microsoft.com/internet/html.htm>

Adding a Background Sound

You can have a background sound play while your Web pages are open by using the `<BGSOUND>` tag in your document. `<BGSOUND>` takes the `SRC` attribute, which is set equal to the URL of a file containing the sound. The file can be in .Wav, .Au, or .Mid (MIDI) format.

`<BGSOUND>` also takes the `LOOP` attribute, which lets you specify how many times to play the sound. `LOOP` can be set to a specific number of times to repeat the sound or to `INFINITE` to play the sound as long as the page is open. For example:

```
<BGSOUND SRC="greeting.wav" LOOP=3>
```

prompts Internet Explorer to deliver your greeting three times when the page is opened. The following HTML:

```
<BGSOUND SRC="greeting.wav" LOOP=INFINITE>
```

causes Internet Explorer to deliver your greeting as long as the page is open.

TIP

Keep your sound files small. Large sound files can take a long time to download- users may be off your page before the sound has finished downloading.

CAUTION

Watch out for the "It's A Small World After All" effect. By looping a sound infinitely, it will play as long as the page is open. Users who find this annoying-just like some do with the famous ride at Disney World-are less likely to visit your pages again in the future.

Creating a Watermark

You can use an image as the background of your documents by using the `BACKGROUND` attribute of the `<BODY>` tag. `BACKGROUND` is set equal to the URL of the image to be used. If the image is not big enough to fit the entire screen, it will be tiled (both horizontally and vertically) to fill the available space.

If you use the `BACKGROUND` attribute of the `<BODY>` tag to tile a graphic as your document background, the background scrolls as you scroll through the document. Internet Explorer gives you greater control over scrolling by supporting a `BGPROPERTIES` attribute. If you set `BGPROPERTIES` to `FIXED`, the background image will not scroll as you move through the document, creating a "watermark" effect.

Controlling Your Margins

Internet Explorer supports the `LEFTMARGIN` and `TOPMARGIN` attributes of the `<BODY>` tag. You can set either one to the number of pixels of white space you want Internet Explorer to leave along the left and top edges of the browser window. Adding some space at the margins often enhances the readability of your documents (see Figure 16.3).

NOTE

You can set both `LEFTMARGIN` and `TOPMARGIN` to 0 to bring both margins right to the edge of the Internet Explorer window.

Adding Color

Internet Explorer 3 supports all of the usual attributes having to do with color (`BGCOLOR`, `LINK`, `VLINK`, and `ALINK`), plus a few others that are often useful. Specifically, with Internet Explorer 3, you can paint table borders, table cells, and horizontal rule with the color of your choice.

Table Backgrounds and Borders

In the `<TABLE>` tag, you can specify the `BORDERCOLOR` attribute to control which color Internet Explorer uses when rendering table borders. `BORDERCOLOR` should be set equal to the hexadecimal RGB triplet or an English language color name that describes the desired color.

`BORDERCOLOR` is useful when rendering a table on a light background. Often, there is not a sharp contrast between the background and the table border when the background is white or a light shade of gray. By darkening the border, you create greater contrast and make the table easier to read.

If you want to use the shaded rule to achieve a three-dimensional effect, then you can control the two colors used to create the shading with `BORDERCOLORDARK` and `BORDERCOLORLIGHT`. Each can be set to an RGB hexadecimal triplet or to an English language color name.

NOTE

You can use the `BORDERCOLOR`, `BORDERCOLORDARK`, and `BORDERCOLORLIGHT` attributes in `<TD>`, `<TH>`, and `<TR>` tags as well as in the `<TABLE>` tag.

You can also use the `BGCOLOR` attribute in a `<TABLE>`, `<TD>`, `<TH>`, or `<TR>` tag to change the background color of a table, table cell, or table row, respectively. `BGCOLOR` is also set equal to a hexadecimal RGB triplet or an English language color name.

Microsoft makes good use of colored table cells on its site. By turning off the borders in the table, the cell looks like a colored, rectangular block floating on the page, and content in the floating block stands out nicely.

CAUTION

Be careful about coloring each of your table cells with different colors. Depending on the combination of colors you use, this can be very harsh on a reader's eyes.

Rule

Microsoft added the `COLOR` attribute to the `<HR>` tag to give HTML authors control over the color in which horizontal rule is rendered. You can set color equal to any RGB hexadecimal triplet or one of the English language color names approved in HTML 3.2.

Being able to color rule helps with contrast in much the same way that `BORDERCOLOR` does. The default

rule has shading behind it, making it difficult to see on light backgrounds. The `NOSHADE` attribute helps with this somewhat, but the color of the rule may still not mesh well with the color scheme of the page. By using `NOSHADE` and `COLOR` together, you can get a solid line in whatever color you choose.

Table Sections and Column Properties

Microsoft has added several tags to the set of table tags that allow you to split tables into logical sections control alignment properties of columns of data, and control data row alignment properties.

Table Sections

The `<THEAD>`, `<TBODY>`, and `<TFOOT>` container tags denote the start of a table header, body, and footer, respectively. By explicitly distinguishing the different parts of your table, you're better able to control how column attributes are applied.

`<THEAD>` contains the rows that comprise the table header and `<TFOOT>` contains the rows that comprise the footer. In the absence of `<THEAD>` and `<TFOOT>` tags, the `<TBODY>` tag becomes optional. You can use multiple `<TBODY>` tags in long tables to make smaller, more manageable chunks.

NOTE
All three tags are only valid between the <code><TABLE></code> and <code></TABLE></code> tags.

A typical table done with these tags might look like:

```
<TABLE>
  <THEAD>
    <TR>
      ...
    </TR>
  </THEAD>
  <TBODY>
    <TR>
      ...
    </TR>
    <TR>
      ...
    </TR>
    ...
    <TR>
      ...
    </TR>
  </TBODY>
  <TFOOT>
    <TR>
      ...
    </TR>
  </TFOOT>
</TABLE>
```

NOTE
According to Microsoft, use of the <code></THEAD></code> , <code></TBODY></code> , and <code></TFOOT></code> tags is optional.

Used in conjunction with the column grouping tags discussed in the next section, the table section tags are an ideal way to control how different properties are applied to different parts of a table.

Setting Column Properties

The `<TR>` tag supports attributes that allow you to specify all sorts of properties for an entire row of a

table. In particular, you get very good control over both horizontal and vertical alignment with the `ALIGN` and `VALIGN` attributes. Microsoft has taken this a step further by making it possible to apply horizontal alignment properties to *columns* of data as well as rows.

You have two options when applying alignment properties to columns. The `<COLGROUP>` tag is appropriate when applying properties over several columns. It takes the attributes `ALIGN`, which can be set to `LEFT`, `CENTER`, or `RIGHT`, and `SPAN`, which is set to the number of consecutive columns that the properties apply to. In the following example:

```
<TABLE>
  <COLGROUP ALIGN=RIGHT SPAN=2>
  <COLGROUP ALIGN=CENTER>
  <COLGROUP ALIGN=LEFT SPAN=3>
  <TBODY>
    <TR>
      <TD>First column group, right aligned</TD>
      <TD>First column group, right aligned</TD>
      <TD>Second column group, center aligned</TD>
      <TD>Third column group, left aligned</TD>
      <TD>Third column group, left aligned</TD>
      <TD>Third column group, left aligned</TD>
    </TR>
  </TBODY>
</TABLE>
```

the six columns are split into three groups. The first two columns have table entries right aligned, the third column's entries are centered, and the last three columns have entries left aligned.

If columns in a group are to have differing properties, you can use `<COLGROUP>` to set up the group and then specify the individual properties with the `<COL>` tag. `<COL>` takes the same attributes as `<COLGROUP>`, but these attributes only apply to a subset of the columns in a group. For example, the HTML:

```
<TABLE>
  <COLGROUP>
    <COL ALIGN=CENTER>
    <COL ALIGN=RIGHT>
  <COLGROUP>
    <COL ALIGN=CENTER SPAN=2>
  <TBODY>
    <TR>
      <TD>First column in first group, center aligned</TD>
      <TD>Second column in first group, right aligned</TD>
      <TD>First column in second group, center aligned</TD>
      <TD>Second column in second group, center aligned</TD>
    </TR>
  </TBODY>
</TABLE>
```

splits the four columns of the table into two groups of two columns each. The first group's columns use center and right alignments, while both columns in the second group use center alignment.

NOTE

Microsoft says that `</COLGROUP>` and `</COL>` closing tags are not required. However, you may want to use them for their organizational value.

Other Attributes of the `<TABLE>` Tag

Microsoft adds two other new attributes to the `<TABLE>` tag that let you control inner and outer borders of a table. Inner borders are controlled by the `RULES` attribute. You can think of inner borders as the

dividing lines between certain components of the table. RULES can take on the values shown in Table 16.1.

Table 16.1 Values of the RULES Attribute of the <TABLE> Tag

Value	Purpose
ALL	Display a border between all rows and columns
COLS	Display a border between all columns
GROUPS	Display a border between all logical groups (as defined by the <THEAD>, <TBODY>, <TFOOT>, and <COLGROUP> tags)
NONE	Suppress all inner borders
ROWS	Display a border between all table rows

The FRAMES attribute controls which sides of the outer borders are displayed.

Table 16.2 Values of the FRAMES Attribute of the <TABLE> Tag

Value	Purpose
ABOVE	Displays a border on the top of a table frame
BELOW	Displays a border at the bottom of a table frame
BORDER	Displays a border on all four sides of a table frame
BOX	Same as BORDER
HSIDES	Displays a border on the left and right sides of a table frame
LHS	Displays a border on the left-hand side of a table frame
RHS	Displays a border on the right-hand side of a table frame
VSIDES	Displays a border at the top and bottom of a table frame
VOID	Suppresses the display of all table frame borders

Inline AVI Support

AVI, short for *Audio Video Interleave*, is the video format Microsoft has designed for use on Windows platforms. For this reason, it is also referred to as Video for Windows. AVI ranks right up there with QuickTime and MPEG as a popular video format for World Wide Web sites.

Microsoft has greatly extended the tag to provide exceptional support of inline video clips and VRML worlds stored in the AVI format. The extended attributes are shown in Table 16.3.

Table 16.3 Internet Explorer Extensions to the Tag

Attribute	Purpose
DYNSRC="url"	Specifies the URL of the AVI file containing the video clip
CONTROLS	Places a control panel in the browser window so the user can control the playing of the clip
START=FILEOPEN MOUSEOVER	Specifies when to start the video clip
LOOP= <i>n</i> INFINITE	Controls how many times the clip is repeated
LOOPDELAY= <i>n</i>	Specifies how many milliseconds to wait before repeating the clip

For example, the following HTML tag:

```
<IMG DYNSRC="leno.avi" CONTROLS START=FILEOPEN LOOP=2>
```

instructs Internet Explorer to play the clip stored in `leno.avi` two times when the file is opened. A control panel will be present while the clip is playing, so that the user may stop, rewind, or fast-forward.

TIP

You can set START both FILEOPEN and MOUSEOVER together (START=FILEOPEN, MOUSEOVER). This configuration will play the clip once when the file is opened, and once each time the mouse is moved over the clip window.

Displaying All Video Formats Inline with ActiveMovie

Microsoft unveiled its ActiveMovie technology along with the second beta release of Internet Explorer 3. Once installed, ActiveMovie enables Internet Explorer to display all three major Web video formats-QuickTime, MPEG, and AVI-inline. This spares the user from having to install helper applications for each format and makes viewing Web video much more convenient (except for the long download times).

In addition to the major video formats, ActiveMovie also supports the major Web audio formats, including .Au, .Wav, MIDI (.Mid), and .Aiff.

Microsoft expects ActiveMovie to become the video standard for both video and desktop applications. To learn more about ActiveMovie Technology, direct your browser to <http://www.microsoft.com/imedia/>.

Scrolling Marquees

The <MARQUEE> and </MARQUEE> tag pair places a scrolling text marquee on your Web page. The text that scrolls is the text found between the two tags.

The <MARQUEE> tag can take a number of attributes that give you very fine control over the appearance and behavior of the marquee. These attributes are summarized in Table 16.4.

Table 16.4 Attributes of the <MARQUEE> Tag

Attribute	Purpose
BGCOLOR="RGB triplet "	Specifies the background color of the marquee window
BEHAVIOR=SCROLL SLIDE ALTERNATE	Specifies how the text should move in the marquee window
DIRECTION=LEFT RIGHT	Controls the direction in which the marquee text moves
SCROLLAMOUNT= <i>n</i>	Sets the number of pixels of space between successive presentations of marquee text
SCROLLDELAY= <i>n</i>	Sets the number of milliseconds to wait before repeating the marquee text
HEIGHT= <i>pixels</i> <i>percent</i>	Specifies the height of the marquee window in either pixels or a percentage of the browser window height
WIDTH= <i>pixels</i> <i>percent</i>	Specifies the width of the marquee window in either pixels or a percentage of the browser window width
HSPACE= <i>n</i>	Specifies how many pixels to make the left and right margins of the marquee window
VSPACE= <i>n</i>	Specifies how many pixels to make the top and bottom margins of the marquee window
LOOP= <i>n</i> INFINITE	Controls how many times the marquee text should scroll
ALIGN=TOP MIDDLE BOTTOM	Specifies how text outside the marquee window should be aligned with the window

Notice in Table 16.4 that many of the attributes are the same as for the tag and that those attributes work in a similar way. HEIGHT and WIDTH define the dimensions of the marquee, HSPACE and VSPACE govern the space around the marquee, and ALIGN controls where subsequent text appears relative to the marquee window.

Of the remaining attributes, **BEHAVIOR** requires some explanation. Setting **BEHAVIOR** to **SCROLL** makes the marquee text scroll on, and then off, the marquee window in the direction specified by the **DIRECTION** attribute. If **BEHAVIOR** is set to **SLIDE**, the text will slide into the window and stay there. If **BEHAVIOR** is set to **ALTERNATE**, the text will bounce back and forth in the window.

Marquee text is a nice Web page effect that you can accomplish on other browsers only by using something more advanced, such as Java or Shockwave. Figure 16.8 shows some marquee text sliding onto a page.

Floating Frames

Microsoft introduced the concept of a *floating frame* with Internet Explorer 3. You can think of a floating frame as a smaller browser window that you can open in your main browser window-much like the "picture-in-a-picture" feature that comes with many television sets. Just as with regular frames, you can load any HTML document you want into a floating frame. The primary difference is that floating frames can be placed anywhere on a page that you can place an image. In fact, you'll find the HTML syntax for placing floating frames to be very similar to that for placing an image.

You place a floating frame on a page by using the `<IFRAME>` and `</IFRAME>` tags. Internet Explorer will ignore anything between these two tags, allowing you to place an alternative to the floating frame (most likely text or an image) on the page as well. This way, browsers that don't know how to render floating frames can ignore the `<IFRAME>` and `</IFRAME>` tags and act on what is found between them. The `<IFRAME>` tag can take the attributes summarized in Table 16.5.

The `<IFRAME>` tag has three required attributes: **WIDTH**, **HEIGHT**, and **SRC**. **WIDTH** and **HEIGHT** specify the width and height of the floating frame in pixels or as a percentage of the browser screen's width and height. **SRC** tells the browser the URL of the document to load into the floating frame. Thus, your basic floating frame HTML looks like:

```
<IFRAME WIDTH=250 HEIGHT=112  
SRC="http://www.your_firm.com/floating.html">  
Text or image based alternative to the floating frame  
</IFRAME>
```

Table 16.5 Attributes of the `<IFRAME>` Tag

Attribute	Purpose
<code>ALIGN=LEFT RIGHT</code>	Floats the floating frame in the left or right margin
<code>FRAMEBORDER=0 1</code>	Controls the presence of the beveled border around the floating frame
<code>HEIGHT=pixels percent</code>	Specifies the height of the floating frame
<code>HSPACE=n</code>	Specifies how many pixels of white space to leave to the left and right of the floating frame
<code>NAME="frame_name"</code>	Gives the floating frame a unique name so it can be targeted by hyperlinks
<code>SCROLLING=YES NO</code>	Controls the presence of scrollbars on the floating frame
<code>SRC="url"</code>	Specifies the URL of the document to load into the floating frame
<code>VSPACE=n</code>	Specifies how many pixels of white space to leave above and below the floating frame
<code>WIDTH=pixels percent</code>	Specifies the width of the floating frame

In addition to the three required attributes, the `<IFRAME>` tag takes several other attributes that give you good control over the floating frame's appearance. These include:

- **FRAMEBORDER** You'll notice in Figure 16.9 that the floating frame has a beveled border that gives it the appearance of being recessed in the main browser window. If you prefer a more seamless look, you can use the **FRAMEBORDER** attribute in the `<IFRAME>` tag. Setting **FRAMEBORDER=0** eliminates the beveled border.

- **SCROLLING** Internet Explorer will put a scrollbar on the floating frame if the document in the frame exceeds the dimensions of the frame. You can suppress the scrollbars by specifying **SCROLLING=NO** in the **<IFRAME>** tag. If you always want scrollbars present, you can set **SCROLLING** equal to **YES**.
- **HSPACE** and **VSPACE** If your floating frame needs some clear space around it, the **HSPACE** and **VSPACE** attributes of the **<IFRAME>** tag work the same way they do for the **** tag: **HSPACE** adds clear space to the left and right of the floating frame, and **VSPACE** adds clear space above and below. **HSPACE** and **VSPACE** values are in pixels.
- **ALIGN** You can float the floating frame in the left or right margins by specifying **ALIGN=LEFT** or **ALIGN=RIGHT**. Any text following the floated frame will wrap around it to the right or left. You can use the **
** tag with the appropriate **CLEAR** attribute to break to the first line clear of floated frames.
- **NAME** Naming a floating frame allows you to target it with the **TARGET** attribute in an **<A>** tag. This allows you to set up links to documents and have them appear in the floating frame.

HTML Layout Control

While not technically part of HTML, Microsoft has introduced HTML Layout Control as one of its ActiveX controls. The HTML Layout Control provides you with fully two-dimensional layout capabilities on a Web page, including the ability to

- Create fixed layout regions with precise control over placement of page elements in the region.
- Overlap images, including transparent images.
- Build mouse rollover effects and pop-up regions into a page.

Previously, this type of functionality was only available by using programming language like Java or a Macromedia Director movie viewed with the Shockwave plug-in. HTML Layout Control makes all of this possible with little-to-no knowledge of programming required on your part.

The key to using HTML Layout Control is the ActiveX Control Pad, which provides a WYSIWYG environment for creating two-dimensional regions with these special effects. Inside the Control Pad window, you can type ordinary HTML code, design a fixed layout region, or embed either VBScript or JavaScript code with the Script Wizard. All of these elements combine to produce Web pages with rich multimedia content that is highly interactive and positioned precisely where the author wants it

NOTE

To download the ActiveX Control Pad, direct your browser (presumably Internet Explorer) to <http://www.microsoft.com/workshop/author/cpad/download.htm>.

Apéndice D: Netscape Navigator-Specific HTML Extensions

CONTENTS

- [A Word About Browser-Specific Extensions](#)
 - [Should You Use Them?](#)
 - [Providing Alternatives](#)
- [Embedded Objects](#)
- [Dynamic Documents](#)
 - [Client Pull](#)
 - [Server Push](#)
- [Multicolumn Text Layout](#)
- [Controlling White Space](#)
 - [HORIZONTAL and VERTICAL Spacers](#)
 - [BLOCK Spacers](#)
- [Frame Border Controls](#)
- [Text Wrapping in Multiple Line Form Windows](#)
- [Text Effects](#)
 - [Changing the Default Font Size](#)
 - [Blinking Text](#)
 - [Strikethrough and Underline Text](#)
 - [Choosing Typefaces](#)

Netscape Communications Corporation was the first browser software company to introduce extensions to HTML—new tags and attributes to existing tags that only Netscape Navigator could render properly. The nature of these extensions was typically driven by design considerations, since the original releases of HTML provided little support for control over the on-screen layout of a document. Netscape's extended version of HTML gave content authors several new options for creating attractive layouts. End users loved the pages they saw that were "Best Viewed with Netscape Navigator," and Navigator quickly became the browser of choice.

In the time since Netscape introduced its first extensions, other companies—Microsoft, in particular—have created proprietary HTML tags that only their browsers can process. As you design and author HTML documents, you need to at least be aware of these extensions and, if appropriate, use them on your pages. This chapter investigates the Netscape-specific extensions to HTML. Microsoft-specific extensions are covered in [Chapter 16](#), "Internet Explorer-Specific HTML Extensions."

A Word About Browser-Specific Extensions

Browser-specific HTML extensions have created quite a fervor in the Web community, not all of it good. From a business standpoint, introducing extensions was an incredibly smart move on Netscape's part. Netscape's objective was to get as many people as possible to use its browser. What better way than to make it the only browser that can be used to view really cool Web pages? By introducing extensions to standard HTML, Netscape achieved its goal and became the most popular browser in use.

From the perspective of open standards, however, extensions to HTML are a nightmare. The notion of open standards suggests that all members of the Internet community can propose ideas for how to do something, and, if the appropriate standards body agrees, the idea is then made part of the accepted standard. While Netscape and other companies typically submit their ideas for extended HTML tags to the World Wide Web Consortium, the standards body for HTML, they don't wait for the W3C's approval before implementing the extensions. Instead, they implement them, immediately hoping to gain an advantage by being "first to market." This leaves the Web community with different browsers that have different capabilities, which, in turn, creates difficulties for content providers who are left unsure which browser they should author pages for.

Another take on extended HTML comes from the HTML purists of the world who believe that HTML is a document description tool and *not* a design tool. For adherents to this philosophy, extensions to standard HTML for the purpose of giving authors more control over presentation is the greatest of offenses. Fortunately, this battle appears to be settling down somewhat now that style sheets are emerging onto the Web scene. As you read in the last chapter, style sheets divorce content from presentation by storing the presentation characteristics separately. This leaves HTML free to just describe the content.

NOTE

Similar philosophical battles are taking place with other document description languages as well. For example, Hyper-G stores link information separately, rather than as part of the content. This approach has its proponents and opponents, just as treating HTML as a design tool does.

Should You Use Them?

With all the controversy surrounding extensions to HTML, you may find yourself asking whether you should use them or not. Ultimately, your answer to this query should be guided by your knowledge of the user community. If you're certain that a large percentage of your audience is using Microsoft Internet Explorer, then there's probably little harm in using Microsoft-specific extensions to HTML in your documents. If your audience is using a mix of browsers, you should consider sticking with standard HTML so that every user can see your pages in the same way that any other user sees them.

Providing Alternatives

If you do use extended HTML tags, and you're not certain that 100% of your audience is using a browser that can correctly parse those tags, you should provide some type of alternative to the effect created by the extended tag so that viewers with less capable browsers can at least read your content. The idea is similar to using the ALT attribute in an tag so that users with text-only browsers, or with image loading turned off, can see a text-based description of the image they would be able to see with a different browser.

Thankfully, most of the extended attributes include some means of providing an alternative. In some cases, as with frames, there is a specific tag you can use to create the alternative. For frames, you can use the <NOFRAMES> and </NOFRAMES> tags to contain a non-framed version of a framed document. In other cases, the extended tag will degrade into something that a less capable browser can understand. An example of this is the proposed <FIG> tag for

placing figures. <FIG> and its companion closing tag </FIG> can contain HTML that is rendered if the figure cannot be displayed. A browser that doesn't understand <FIG> and </FIG> will ignore these tags, but it will process and render what it finds between them. Thus, you can place your alternative to the <FIG> tag between <FIG> and </FIG>.

Providing alternatives to extended HTML tags helps to maximize your reach when your entire audience is not using the same browser. Make sure you use them whenever you employ browser-specific extensions on your pages.

Embedded Objects

Netscape introduced the <EMBED> tag for placing multimedia content on Web pages. <EMBED> takes the SRC attribute, and SRC is set equal to the URL of the file that contains the multimedia content. You also need to specify WIDTH and HEIGHT attributes (in pixels) so that the browser knows how much space to leave on the page for the embedded content.

You can use <EMBED> to place many kinds of items on a page, including:

- **Macromedia Director movies** Users with the Shockwave plug-in to Netscape are able to view Macromedia Director movies inline. The HTML to place a Director movie might look like:

```
<EMBED SRC="my_movie.dcr" WIDTH=278 HEIGHT=225>
```

- **VRML worlds** The Virtual Reality Modeling Language (VRML, rhymes with "thermal") allows you to create three-dimensional objects and place them in a VRML world that users can move through. Netscape Navigator 3.0 comes with a Live3D feature that permits inline viewing of VRML world placed on a page with an <EMBED> tag.

```
<EMBED SRC="jod6.wrl" WIDTH=400 HEIGHT=250>
```

- **Video content** Netscape 3.0 has built-in support for the display of QuickTime and Video for Windows files. Figure 15.3 shows a QuickTime movie playing in a Web page. The movie was placed on the page with the HTML:

```
<EMBED SRC="./italy/audio/giorno.mov" WIDTH=148  
HEIGHT=192>
```

NOTE

For QuickTime movies, you can add a number of other attributes to the <EMBED> tag to control how the movie plays. For more details, consult

http://home.netscape.com/comprod/products/navigator/version_3.0/multimedia/quicktime/how.html

You can also use the <EMBED> tag to place other multimedia content as well. In some instances, Netscape will not be able to display your embedded content on its own. For these files, you should include a notice on the page informing the users what plug-in is required and where they can get it.

Dynamic Documents

Netscape pioneered the idea of *dynamic documents*-documents that change without any prompting from the user. You can create a dynamic document in one of two ways:

- **Client Pull** In a Client Pull, the browser either reloads the current page or loads an

entirely new page after a specified delay.

- **Server Push** A Server Push involves the server sending new content through an HTTP connection that is held open. New content replaces old content on the browser screen, giving the appearance of a live update to the page.

Each of these approaches is discussed in the following sections.

Client Pull

Netscape extended the `<META>` tag in the document head to include a value of "Refresh" for the `HTTP-EQUIV` attribute. `Refresh` instructs the browser to reload the same document, or a different document, after a specified number of seconds. The time delay and the URL of the next document, if applicable, are stored in the `CONTENT` attribute. The syntax for the `<META>` tag in this situation is

```
<META HTTP-EQUIV="Refresh" CONTENT="n; url">
```

where *n* is the number of seconds to wait and *url* is the URL of the next document to load. If you want to reload the same document, just use `CONTENT="n"` with no URL specified.

CAUTION

URLs in the `CONTENT` attribute should be fully qualified (that is, no relative URLs).

Server Push

Server Push is the other technique for creating dynamic documents. With a Server Push, the server keeps the HTTP connection open between it and the client and periodically sends new data to the client through the open connection. The sending of new data continues until the client interrupts the connection, or until the server has no more data to send, at which point it closes the connection.

NOTE

The open HTTP connection is unique to Server Push. With Client Pull, a new connection is opened each time new data is reloaded. You can argue that this makes Client Pull somewhat less efficient, since it can take a comparatively long time to open a new connection. On the other hand, you can argue that Server Push is less efficient because keeping an open HTTP connection wastes a server resource.

Unlike Client Pull, you cannot implement a Server Push with HTML tags. The Server Push is controlled by a script or a program running on the server. The script or program specifies what data to send to the client and when to send it.

The key to sending multiple data sets through the open connection is to create an HTTP response of MIME type `multipart/x-mixed-replace`, where

- `multipart` means that the response contains multiple parts.
- `x` means that the MIME type is experimental.
- `mixed` means that the different parts of the response could be of different MIME types.
- `replace` means that each new piece of data should replace the one that was sent before it.

MIME Types

Multipurpose Internet Mail Extensions (MIME) types are labels that

describe the contents of a certain class of file. For example, HTML documents are of type `text/html` and QuickTime video clips are of type `video/quicktime`. Types let mail and Web servers know what kind of files they're transferring so that they can do so without trashing them.

An experimental MIME type is one that has not undergone review by the Internet community so that it can have a standard implementation. While their usage is generally discouraged, experimental MIME types can be submitted to the Internet Assigned Numbers Authority (IANA) for inclusion in the central MIME type registry. For more information on submitting a new MIME type, see http://www.oac.uci.edu/indiv/ehood/MIME/1521/Appendix_E.html.

The replace feature is critical to Server Pushes. By replacing the previous piece of data right where it was on the screen, you're able to create an animation effect. In fact, prior to the onset of animated GIFs, the most popular application of Server Push dynamic documents was to create inline animations on a Web page. To create a Server Push animation, your script or program needs to set up a `multipart/x-mixed-replace` response in which each part of the response is a frame in the animation. As new frames are sent, they replace the old frames and create the animation!

While Server Pushes came to prominence as a way to create Web page animations, they are useful in other situations as well. Another common use of a Server Push is to send periodically updated information to a page. For example, a page with a stock ticker could receive the ticker symbols and latest prices by Server Push and display them as they roll in.

Multicolumn Text Layout

A new addition to Netscape's collection of extended HTML tags is the `<MULTICOL>` container tag used to create a multiple column layout similar to what you see in a printed newspaper. Prior to the `<MULTICOL>` tag, such layouts were possible with tables, but it was difficult to control column widths and to get each column to be about the same height.

The `<MULTICOL>` tag takes the three attributes shown in Table 15.1. `COLS` is the only mandatory attribute and is set equal to the number of columns that should comprise the layout. By default, Netscape will place a 10-pixel gutter between each column, but you can change this value by using the `GUTTER` attribute. `GUTTER` can be set to a specific number of pixels or to a percentage of the browser screen width. `WIDTH` controls how wide the columns are. Like `GUTTER`, `WIDTH` can be set equal to a number of pixels or a screen width percentage.

CAUTION

Make sure your columns are wide enough to accommodate your widest nonbreaking element. If an element can't be broken and it exceeds the width of the column, it will overlap into adjacent columns.

Table 15.1 Attributes of the `<MULTICOL>` Tag

Attribute	Purpose
<code>COLS=<i>n</i></code>	Makes the layout <i>n</i> columns wide
<code>GUTTER=pixels percentage</code>	Controls the amount of space left between columns
<code>WIDTH=pixels percentage</code>	Controls the width of columns in the layout

When laying out a multicolumn page, Netscape Navigator attempts to make each column the same height. This is a fairly simple matter if the columns contain only text, but the Navigator even does well with columns that have floating images with text wrapping around them.

You can nest multiple column layouts as well by placing `<MULTICOL>` tags inside one another.

```
<MULTICOL COLS=2 GUTTER=20 WIDTH=100%>
<P><TT><B>MULTICOL</B></TT> - The <TT>MULTICOL</TT> tag is
a container,
and all the HTML between the starting and ending tag will
be displayed
in a multicolumn format. The tag can be nested. The
attributes of this
tag are <TT>COLS</TT>, <TT>GUTTER</TT>, and
<TT>WIDTH</TT>. </P>
...
<P><B>Column widths</B> - Column sizes are not adjusted
for optimal fit
the way table cells are. If you specify very narrow
columns and put a
very wide unbreakable element in the column, it will
overlap
neighboring columns.</P>
<P><B>Nested Multicolumns</B> - This is an example of text
in a
nested multicolumn:
<MULTICOL COLS=3 GUTTER=8 WIDTH=100%>
Here, we've entered another <TT>MULTICOL</TT> tag into the
HTML&nbsp;
to produce with a narrow gutter (8 pixels). It is possible
to nest
<TT>MULTICOL</TT> tags infinitely.
</MULTICOL>
<DL><DT>
<B>E<FONT SIZE=-1>XAMPLES</FONT></B><DD>
<A HREF="JavaScript:makeWindow()">View</A> the Investor
Newsletter example.
</DL>
</MULTICOL>
```

You'll notice in Figure 15.8 that you can use any of the usual text formatting tags to mark up text in a multicolumn layout. This is different from tables where you can turn on an effect (like boldface) before you start the table, but the effect is not applied to elements inside the table.

Controlling White Space

Another new tag that was rolled out with Netscape Navigator 3.0 is the `<SPACER>` tag. `<SPACER>` is a standalone tag used to place white space on a page. Table 15.2 shows the attributes you can use with `<SPACER>`.

Table 15.2 Attributes of the `<SPACER>` Tag

Attribute	Purpose
ALIGN=LEFT RIGHT	Floats the spacer in the left or right margin, allowing text to wrap around it
HEIGHT=pixels percentage	Specifies the height of a BLOCK spacer
SIZE=pixels	Specifies the size of a HORIZONTAL or VERTICAL spacer
TYPE=BLOCK HORIZONTAL VERTICAL	Controls whether the spacer adds space horizontally, vertically or both (BLOCK)
WIDTH=pixels percentage	Specifies the width of a BLOCK spacer

HORIZONTAL and VERTICAL Spacers

HORIZONTAL and VERTICAL spacers add white space in the horizontal or vertical directions

only. Spacers of this type require the `SIZE` attribute to specify how many pixels wide or deep they should be.

One clever use of `HORIZONTAL` spacers is to create an indent at the start of a paragraph (see Figure 15.9). Previously, this had to be done with several non-breaking space characters (` `) or with an "invisible image" (an image that is only one color that is set to be transparent).

```
<SPACER TYPE="HORIZONTAL" SIZE=50>
```

Vertical spacers are useful for placing additional space between block elements like paragraphs, horizontal rule, and images.

BLOCK Spacers

`BLOCK` spacers add white space in both the horizontal and vertical directions. When creating a `BLOCK` spacer, you need to use the `WIDTH` and `HEIGHT` attributes to specify the displacements in the two directions. You can also use the `ALIGN` attribute to float the `BLOCK` spacer in the left or right margin.

You can think of a `BLOCK` spacer as a large transparent image of the dimensions you specify in the tag. The tag to create the spacer is

```
<SPACER TYPE="BLOCK" WIDTH= HEIGHT= ALIGN="RIGHT">
```

Frame Border Controls

Netscape has added a few attributes to its frame-related tags to give you greater control over borders between frames. The first is the `FRAMEBORDER` attribute of the `<FRAMESET>` and `<FRAME>` tags. `FRAMEBORDER` has a default value of `YES`, and you can set it to a value of `NO`, if desired.

Inside a `<FRAMESET>` tag, `FRAMEBORDER` governs the default border value for all frames in the frameset. If you set `FRAMEBORDER` to `NO` and `BORDER` to `0`, there will be no borders between any of the frames in the frameset. This gives a "seamless" appearance to the framed layout.

NOTE

The `BORDER` attribute can only be used in your outermost `<FRAMESET>` tag. It controls the width of all borders within the frameset.

When used in a `<FRAME>` tag, `FRAMEBORDER` controls the presence of borders for the frame that the `<FRAME>` tag sets up. Using `FRAMEBORDER` at the `<FRAME>` tag level will override any other border attributes set in a previous `<FRAMESET>` tag.

The other new attribute is `BORDERCOLOR`. `BORDERCOLOR` can also be an attribute to both the `<FRAMESET>` and `<FRAME>` tags. You set `BORDERCOLOR` to an RGB hexadecimal triplet or an English-language color name. Using `BORDERCOLOR` in the `<FRAMESET>` tag sets the color for all frames in the frameset, while using it in a `<FRAME>` tag just sets that color for the frame defined by the tag.

CAUTION

Since frame borders are shared between two frames, you need to be careful about two different color choices for the same border. In a conflict situation, the color in the outermost `<FRAMESET>` tag has the lowest priority. This color is overridden by any color specified in a nested `<FRAMESET>` tag. Finally, a color specified in a `<FRAME>` tag takes precedence over any specified in any `<FRAMESET>` tag. If two colors

have the same priority, the conflict is left unresolved.

Text Wrapping in Multiple Line Form Windows

When creating a multiple line text input window on an HTML form, you use the `<TEXTAREA>` and `</TEXTAREA>` container tags. One problem users frequently have with these windows is the issue of text wrapping. As users type their input, it's inconvenient for them to have to remember to hit Enter near the end of each line so that the line doesn't scroll off the edge of the window.

Netscape has introduced a new attribute of the `<TEXTAREA>` tag to alleviate this problem. By placing the `WRAP` attribute in a `<TEXTAREA>` tag, you enable automatic wrapping of text within the multiline text window. A window set up with the `WRAP` attribute will have no horizontal scrollbar at the bottom of it (compare Figures 15.11 and 15.12) since it should be unnecessary to scroll left or right in such a window.

Text Effects

The last few extensions discussed in this chapter are used to format text on screen. Most of these extensions are useful, though one of them, the `<BLINK>` tag, has become something of an outcast because it was used to make some really obnoxious pages.

Changing the Default Font Size

The use of the `` tag with the `SIZE` attribute is now part of the HTML 3.2 standard. Recall that you can set `SIZE` equal to a value between 1 (the smallest) and 7 (the largest). You could also set the value to a positive or negative number representing how many sizes above or below the base font size you want to go. The default font size is taken to be 3, so using `` is equivalent to saying `` since 5 is two greater than 3.

Netscape Navigator allows you to reassign the default font size from 3 to a different value using the `<BASEFONT>` tag. `<BASEFONT>` is a stand-alone tag that takes the `SIZE` attribute. In this case, `SIZE` is set to the value, a number between 1 and 7, that should be used as the default size.

Blinking Text

Probably Netscape's most infamous extension to HTML is the `<BLINK>` tag. Any text between `<BLINK>` and its companion closing tag `</BLINK>` will blink on the Netscape Navigator screen.

This may seem innocuous enough, but `<BLINK>` became overused and inappropriately used very quickly. By creating documents with large blocks of blinking text, page authors alienated the end users in their audiences who found it very difficult to read the blinking text. Before long, anyone who used the `<BLINK>` tag was immediately subjected to his or her fair share of criticism.

If you do choose to use the `<BLINK>` tag on your pages, please do your users the favor of thinking twice about whether it is really needed or not. If you decide that it is, keep the amount of blinking text to a minimum-no more than three to five words. Anything more than that is distracting and difficult to read.

Strikethrough and Underline Text

You can create strikethrough text and underlined text by using the `<S>` and `<U>` container tags, respectively (see Figure 15.13). Prior to Netscape Navigator 3.0, strikethrough text was also supported using the `<STRIKE>` and `</STRIKE>` tags. You can continue to use these tags for strikethrough text if you want.

The underline style was not supported by Netscape for some time, since underlined text tended to be confused with hyperlinked text (which is also usually underlined, but rendered in a color different from the rest of the body text). Netscape Navigator now fully supports the `<U>` tag and can render underlined text.

Choosing Typefaces

Beginning with release 3.0, Netscape Navigator now recognizes the `FACE` attribute of the `` tag. `FACE` is set equal to a list of typefaces to use when rendering the text between the `` and `` tags. If the first typeface in the list is not available, the browser tries the second. If the second is unavailable, it moves on to the third, and so on. For example, the following HTML:

```
<FONT FACE="Helvetica,Garamond,Palatino">New  
typeface!</FONT>
```

instructs Netscape to render the text `New typeface!` in Helvetica, if it's available. If it isn't, the browser tries to use Garamond, followed by Palatino.

Using the `FACE` attribute gives you the ability to change the body text from plain old Times Roman or Courier to whichever font you want. Changing typefaces is a great way to break up large sections of text in your documents.

TIP

Try to keep your list of fonts limited to the standard ones found on most PC or Macintosh platforms. These include Helvetica, Arial, Times Roman, Courier, Palatino, and Garamond.

NOTE

Technically, Microsoft was the first to introduce the `FACE` attribute of the `` tag. This isn't a significant matter though, since this attribute will most likely give way to typeface specification via HTML style sheets.

Apéndice D: JavaScript Language Reference

The first part of this reference is organized by object with properties and methods listed by the object they apply to. The second part covers independent functions in JavaScript not connected with a particular object, as well as operators in JavaScript.

The anchor Object

See the anchors property of the document object.

The button Object

The button object reflects a push button from an HTML form in JavaScript.

Properties

- **name** A string value containing the name of the button element.
- **value** A string value containing the value of the button element.

Methods

- **click()** Emulates the action of clicking on the button.

Event Handlers

- **onClick** Specifies JavaScript code to execute when the button is clicked.

The checkbox Object

The checkbox object makes a checkbox from an HTML form available in JavaScript.

Properties

- **checked** A boolean value indicating if the checkbox element is checked.
- **defaultChecked** A boolean value indicating if the checkbox element was checked by default (i.e. reflects the CHECKED attribute).
- **name** A string value containing the name of the checkbox element.
- **value** A string value containing the value of the checkbox element.

Methods

- **click()** Emulates the action of clicking on the checkbox.

Event Handlers

- **onClick** Specifies JavaScript code to execute when the checkbox is clicked.

The Date Object

The Date object provides mechanisms for working with dates and times in JavaScript. Instances of the object can be created with the syntax

```
newObjectName = new Date(dateInfo)
```

Where *dateInfo* is an optional specification of a particular date and can be one of the following:

"month day, year hours:minutes:seconds"

year, month, day

year, month, day, hours, minutes, seconds

where the later two options represent integer values.

If no *dateInfo* is specified, the new object will represent the current date and time.

Methods

- **getDate()** Returns the day of the month for the current Date object as an integer from 1 to 31.
- **getDay()** Returns the day of the week for the current Date object as an integer from 0 to 6 (where 0 is Sunday, 1 is Monday, etc.).
- **getHours()** Returns the hour from the time in the current Date object as an integer from 0 to 23.
- **getMinutes()** Returns the minutes from the time in the current Date object as an integer from 0 to 59.
- **getMonth()** Returns the month for the current Date object as an integer from 0 to 11 (where 0 is January, 1 is February, etc.).
- **getSeconds()** Returns the seconds from the time in the current Date object as an integer from 0 to 59.

- **getTime()** Returns the time of the current Date object as an integer representing the number of milliseconds since 1 January 1970 at 00:00:00.
- **getTimezoneOffset()** Returns the difference between the local time and GMT as an integer representing the number of minutes.
- **getYear()** Returns the year of the week for the current Date object as a two-digit integer representing the year less 1900.
- **parse(dateString)** Returns the number of milliseconds between January 1, 1970 at 00:00:00 and the date specified in *dateString*. *dateString* should take the format

Day, DD Mon YYYY HH:MM:SS TZN

Mon DD, YYYY

- **setDate(dateValue)** Sets the day of the month for the current Date object. *dateValue* is an integer from 1 to 31.
- **setHours(hoursValue)** Sets the hours for the time for the current Date object. *hoursValue* is an integer from 0 to 23.
- **setMinutes(minutesValue)** Sets the minutes for the time for the current Date object. *minutesValue* is an integer from 0 to 59.
- **setMonth(monthValue)** Sets the month for the current Date object. *monthValue* is an integer from 0 to 11 (where 0 is January, 1 is February, etc.).
- **setSeconds(secondsValue)** Sets the seconds for the time for the current Date object. *secondsValue* is an integer from 0 to 59.
- **setTime(timeValue)** Sets the value for the current Date object. *timeValue* is an integer representing the number of milliseconds since January 1, 1970 at 00:00:00.
- **setYear(yearValue)** Sets the year for the current Date object. *yearValue* is an integer greater than 1900.
- **toGMTString()** Returns the value of the current Date object in GMT as a string using Internet conventions in the form

Day, DD Mon YYYY HH:MM:SS GMT

- **toLocaleString()** Returns the value of the current Date object in the local time using local conventions.
- **UTC(yearValue, monthValue, dateValue, hoursValue, minutesValue, secondsValue)** Returns the number of milliseconds since January 1, 1970 at 00:00:00 GMT. *yearValue* is an integer greater than 1900. *monthValue* is an integer from 0 to 11. *dateValue* is an integer from 1 to 31. *hoursValue* is an integer from 0 to 23.

minutesValue and secondsValue are integers from 0 to 59. hoursValue, minutesValue and secondsValue are optional.

The document Object

The document object reflects attributes of an HTML document in JavaScript.

Properties

- **alinkColor** The color of active links as a string or a hexadecimal triplet.
- **anchors** Array of anchor objects in the order they appear in the HTML document. Use anchors.length to get the number of anchors in a document.
- **bgColor** The color of the document's background.
- **cookie** A string value containing cookie values for the current document.
- **fgColor** The color of the document's foreground.
- **forms** Array of form objects in the order the forms appear in the HTML file. Use forms.length to get the number of forms in a document.
- **lastModified** String value containing the last date of modification of the document.
- **linkColor** The color of links as a string or a hexadecimal triplet.
- **links** Array of link objects in the order the hypertext links appear in the HTML document. Use links.length to get the number of links in a document.
- **location** A string containing the URL of the current document.
- **referrer** A string value containing the URL of the calling document when the user follows a link.
- **title** A string containing the title of the current document.
- **vlinkColor** The color of followed links as a string or a hexadecimal triplet.

Methods

- **clear()** Clears the document window.

- **close()** Closes the current output stream.
- **open(mimeType)** Opens a stream which allows `write()` and `writeln()` methods to write to the document window. *mimeType* is an optional string which specifies a document type supported by Navigator or a plug-in (i.e. `text/html`, `image/gif`, etc.).
- **write()** Writes text and HTML to the specified document.
- **writeln()** Writes text and HTML to the specified document followed by a newline character.

The form Object

The form object reflects an HTML form in JavaScript. Each HTML form in a document is reflected by a distinct instance of the form object.

Properties

- **action** A string value specifying the URL which the form data is submitted to.
- **elements** Array of objects for each form element in the order in which they appear in the form.
- **encoding** String containing the MIME encoding of the form as specified in the `ENCTYPE` attribute.
- **method** A string value containing the method of submission of form data to the server.
- **target** A string value containing the name of the window that responses to form submissions are directed to.

Methods

- **submit()** Submits the form.

Event Handlers

- **onSubmit** Specifies JavaScript code to execute when the form is submitted. The code should return a true value to allow the form to be submitted. A false value prevents the form from being submitted.

The frame Object

The frame object reflects a frame window in JavaScript.

Properties

- **frames** An array of objects for each frame in a window. Frames appear in the array in the order in which they appear in the HTML source code.
- **parent** A string indicating the name of window containing the frameset.
- **self** A alternative for the name of the current window.
- **top** An alternative for the name of the top-most window.
- **window** An alternative for the name of the current window.

Methods

- **alert(*message*)** Displays *message* in a dialog box.
- **close()** Closes the window.
- **confirm(*message*)** Displays *message* in a dialog box with OK and CANCEL buttons. Returns true or false based on the button clicked by the user.
- **open(*url,name,features*)** Opens *url* in a window named *name*. If *name* doesn't exist, a new window is created with that name. *features* is an optional string argument containing a list of features for the new window. The feature list contains any of the following name-value pairs separated by commas and without additional spaces:

toolbar=[yes,no,1,0]	Indicates if the window should have a toolbar
location=[yes,no,1,0]	Indicates if the window should have a location field
directories=[yes,no,1,0]	Indicates if the window should have directory buttons
status=[yes,no,1,0]	Indicates if the window should have a status bar
menubar=[yes,no,1,0]	Indicates if the window should have menus
scrollbars=[yes,no,1,0]	Indicates if the window should have scroll bars
resizable=[yes,no,1,0]	Indicates if the window should be resizable
width= <i>pixels</i>	Indicates the width of the window in pixels
height= <i>pixels</i>	Indicates the height of the window in pixels

- **prompt(*message,response*)** Displays *message* in a dialog box with a text entry field with the default value of *response*. The user's response in the text entry field is returned as a string.
- **setTimeout(*expression,time*)** Evaluates *expression* after *time* where *time* is a value in milliseconds. The time out can be named with the structure:

name = `setTimeout(expression,time)`

- **clearTimeout(*name*)** Cancels the time out with the name *name*.

The hidden Object

The hidden object reflects a hidden field from an HTML form in JavaScript.

Properties

- **name** A string value containing the name of the hidden element.
- **value** A string value containing the value of hidden text element.

The history Object

The history object allows a script to work with the Navigator browser's history list in JavaScript. For security and privacy reasons, the actual content of the list is not reflected into JavaScript.

Properties

- **length** An integer representing the number of items on the history list.

Methods

- **back()** Goes back to the previous document in the history list.
- **forward()** Goes forward to the next document in the history list.
- **go(*location*)** Goes to the document in the history list specified by *location*. *location* can be a string or integer value. If it is a string it represents all or part of a URL in the history list. If it is an integer, *location* represents the relative position of the document on the history list. As an integer, *location* can be positive or negative.

The link object

The link object reflects a hypertext link in the body of a document.

Properties

- **target** A string value containing the name of the window or frame specified in the TARGET attribute.

Event Handlers

- **onClick** Specifies JavaScript code to execute when the link is clicked.
- **onMouseOver** Specifies JavaScript code to execute when the mouse is over the hypertext link

The location Object

The location object reflects information about the current URL.

Properties

- **hash** A string value containing the anchor name in the URL.
- **host** A string value containing the hostname and port number from the URL.
- **hostname** A string value containing the domain name (or numerical IP address) from the URL.
- **href** A string value containing the entire URL.
- **pathname** A string value specifying the path portion of the URL.
- **port** A string value containing the port number from the URL.
- **protocol** A string value containing the protocol from the URL (including the colon, but not the slashes).
- **search** A string value containing any information passed to a GET CGI-BIN call (i.e. an information after the question mark).

The Math Object

The Math object provides properties and methods for advanced mathematical calculations.

Properties

- **E** The value of Euler's constant (roughly 2.718) used as the base for natural logarithms.
- **LN10** The value of the natural logarithm of 10 (roughly 2.302).

- **LN2** The value of the natural logarithm of 2 (roughly 0.693).
- **PI** The value of PI—used in calculating the circumference and area of circles (roughly 3.1415).
- **SQRT1_2** The value of the square root of one-half (roughly 0.707).
- **SQRT2** The value of the square root of two (roughly 1.414).

Methods

- **abs(*number*)** Returns the absolute value of *number*. The absolute value is the value of a number with it's sign ignored so `abs(4)` and `abs(-4)` both return 4
- **acos(*number*)** Returns the arc cosine of *number* in radians.
- **asin(*number*)** Returns the arc sine of *number* in radians.
- **atan(*number*)** Returns the arc tangent of *number* in radians.
- **ceil(*number*)** Returns the next integer greater than *number*—in other words, rounds up to the next integer.
- **cos(*number*)** Returns the cosine of *number* where *number* represents an angle in radians.
- **exp(*number*)** Returns the value of E to the power of *number*.
- **floor(*number*)** Returns the next integer less than *number*—in other words, rounds down to the nearest integer.
- **log(*number*)** Returns the natural logarithm of *number*.
- **max(*number1*,*number2*)** Returns the greater of *number1* and *number2*.
- **min(*number1*,*number2*)** Returns the smaller of *number1* and *number2*.
- **pow(*number1*,*number2*)** Returns the value of *number1* to the power of *number2*.
- **random()** Returns a random number between zero and one (at press time, this method only was available on UNIX versions of Navigator 2.0).

- **round(*number*)** Returns the closest integer to *number*—in other words rounds to the closest integer.
- **sin(*number*)** Returns the sine of *number* where *number* represents an angle in radians.
- **sqrt(*number*)** Returns the square root of number.
- **tan(*number*)** Returns the tangent of *number* where *number* represents an angle in radians.

The navigator object

The navigator object reflects information about the version of Navigator being used.

Properties

- **appName** A string value containing the code name of the client (i.e. "Mozilla" for Netscape Navigator).
- **appName** A string value containing the name of the client (i.e. "Netscape" for Netscape Navigator).
- **appVersion** A string value containing the version information for the client in the form *versionNumber (platform; country)*

For instance, Navigator 2.0, beta 6 for Windows 95 (international version), would have an appVersion property with the value "2.0b6 (Win32; I)".

- **userAgent** A string containing the complete value of the user-agent header sent in the HTTP request. This contains all the information in appName and appVersion:
Mozilla/2.0b6 (Win32; I)

The password Object

The password object reflects a password text field from an HTML form in JavaScript.

Properties

- **defaultValue** A string value containing the default value of the password element (i.e. the value of the VALUE attribute).
- **name** A string value containing the name of the password element.

- **value** A string value containing the value of the password element.

Methods

- **focus()** Emulates the action of focusing in the password field.
- **blur()** Emulates the action of removing focus from the password field.
- **select()** Emulates the action of selecting the text in the password field.

The radio Object

The radio object reflects a set of radio buttons from an HTML form in JavaScript. To access individual radio buttons, use numeric indexes starting at zero. For instance, individual buttons in a set of radio buttons named testRadio could be referenced by testRadio[0], testRadio[1], etc.

Properties

- **checked** A boolean value indicating if a specific button is checked. Can be used to select or deselect a button.
- **defaultChecked** A boolean value indicating if a specific button was checked by default (i.e. reflects the CHECKED attribute).
- **length** An integer value indicating the number of radio buttons in the set.
- **name** A string value containing the name of the set of radio buttons.
- **value** A string value containing the value a specific radio button in a set (i.e. reflects the VALUE attribute).

Methods

- **click()** Emulates the action of clicking on a radio button.

Event Handlers

- **onClick** Specifies JavaScript code to execute when a radio button is clicked.

The reset Object

The reset object reflects a reset button from an HTML form in JavaScript.

Properties

- **name** A string value containing the name of the reset element.
- **value** A string value containing the value of the reset element.

Methods

- **click()** Emulates the action of clicking on the reset button.

Event Handlers

- **onClick** Specifies JavaScript code to execute when the reset button is clicked.

The select Object

The select object reflects a selection list from an HTML form in JavaScript.

Properties

- **length** An integer value containing the number of options in the selection list.
- **name** A string value containing the name of the selection list.
- **options** An array reflecting each of the options in the selection list in the order they appear. The options property has it's own properties:

defaultSelected	A boolean value indicating if an option was selected by default (i.e. reflects the SELECTED attribute).
index	An integer value reflecting the index of an option.
length	An integer value reflecting the number of options in the selection list.
name	A string value containing the name of the selection list.
options	A string value containing the full HTML code for the selection list.
selected	A boolean value indicating if the option is selected. Can be used to select or deselect an option.
selectedIndex	An integer value containing the index of the currently selected option.
text	A string value containing the text displayed in the selection list for a particular option.
value	A string value indicating the value for the specified option (i.e. reflects the VALUE attribute).

- **selectedIndex** Reflects the index of the currently selected option in the selection list.

Event Handlers

- **onBlur** Specifies JavaScript code to execute when the selection list loses focus.

- **onFocus** Specifies JavaScript code to execute when focus is given to the selection list.
- **onChange** Specifies JavaScript code to execute when the selected option in the list changes.

The string Object

The string object provides properties and methods for working with string literals and variables.

Properties

- **length** An integer value containing the length of the string expressed as the number of characters in the string.

Methods

- **anchor(*name*)** Returns a string containing the value of the string object surrounded by an A container tag with the NAME attribute set to *name*.
- **big()** Returns a string containing the value of the string object surrounded by a BIG container tag.
- **blink()** Returns a string containing the value of the string object surrounded by a BLINK container tag.
- **bold()** Returns a string containing the value of the string object surrounded by a B container tag.
- **charAt(*index*)** Returns the character at the location specified by *index*.
- **fixed()** Returns a string containing the value of the string object surrounded by a FIXED container tag.
- **fontColor(*color*)** Returns a string containing the value of the string object surrounded by a FONT container tag with the COLOR attribute set to *color* where *color* is a color name or an RGB triplet.
- **fontSize(*size*)** Returns a string containing the value of the string object surrounded by a FONTSIZE container tag with the size set to *size*.
- **indexOf(*findString*,*startingIndex*)** Returns the index of the first occurrence of *findString*, starting the search at *startingIndex* where *startingIndex* is optional—if it is not provided, the search starts at the start of the string.
- **italics()** Returns a string containing the value of the string object surrounded by an I

container tag.

- **lastIndexOf(findString,startingIndex)** Returns the index of the last occurrence of *findString*. This is done by searching backwards from *startingIndex*. *startingIndex* is optional and assumed to be the last character in the string if no value is provided.
- **link(href)** Returns a string containing the value of the string object surrounded by an A container tag with the HREF attribute set to *href*.
- **small()** Returns a string containing the value of the string object surrounded by a SMALL container tag.
- **strike()** Returns a string containing the value of the string object surrounded by a STRIKE container tag.
- **sub()** Returns a string containing the value of the string object surrounded by a SUB container tag.
- **substring(firstIndex,lastIndex)** Returns a string equivalent to the substring starting at *firstIndex* and ending at the character before *lastIndex*. If *firstIndex* is greater than *lastIndex*, the string starts at *lastIndex* and ends at the character before *firstIndex*.
- **sup()** Returns a string containing the value of the string object surrounded by a SUP container tag.
- **toLowerCase()** Returns a string containing the value of the string object with all character converted to lower case.
- **toUpperCase()** Returns a string containing the value of the string object with all character converted to upper case.

The submit Object

The submit object reflects a submit button from an HTML form in JavaScript.

Properties

- **name** A string value containing the name of the submit button element.
- **value** A string value containing the value of the submit button element.

Methods

- **click()** Emulates the action of clicking on the submit button.

Event Handlers

- **onClick** Specifies JavaScript code to execute when the submit button is clicked.

The text Object

The text object reflects a text field from an HTML form in JavaScript.

Properties

- **defaultValue** A string value containing the default value of the text element (i.e. the value of the VALUE attribute).
- **name** A string value containing the name of the text element.
- **value** A string value containing the value of the text element.

Methods

- **focus()** Emulates the action of focusing in the text field.
- **blur()** Emulates the action of removing focus from the text field.
- **select()** Emulates the action of selecting the text in the text field.

Event Handlers

- **onBlur** Specifies JavaScript code to execute when focus is removed from the field.
- **onChange** Specifies JavaScript code to execute when the content of the field is changed.
- **onFocus** Specifies JavaScript code to execute when focus is given to the field.
- **onSelect** Specifies JavaScript code to execute when the user selects some or all of the text in the field.

The textarea Object

The textarea object reflects a multi-line text field from an HTML form in JavaScript.

Properties

- **defaultValue** A string value containing the default value of the textarea element (i.e. the value of the VALUE attribute).
- **name** A string value containing the name of the textarea element.
- **value** A string value containing the value of the textarea element.

Methods

- **focus()** Emulates the action of focusing in the textarea field.
- **blur()** Emulates the action of removing focus from the textarea field.
- **select()** Emulates the action of selecting the text in the textarea field.

Event Handlers

- **onBlur** Specifies JavaScript code to execute when focus is removed from the field.
- **onChange** Specifies JavaScript code to execute when the content of the field is changed.
- **onFocus** Specifies JavaScript code to execute when focus is given to the field.
- **onSelect** Specifies JavaScript code to execute when the user selects some or all of the text in the field.

The window Object

The window object is the top-level object for each window or frame and is the parent object for the document, location and history objects.

Properties

- **defaultStatus** A string value containing the default value displayed in the status bar.
- **frames** An array of objects for each frame in a window. Frames appear in the array in the order in which they appear in the HTML source code.
- **length** An integer value indicating the number of frames in a parent window.

- **name** A string value containing the name of the window or frame.
- **parent** A string indicating the name of the window containing the frameset.
- **self** A alternative for the name of the current window.
- **status** Used to display a message in the status bar—this is done by assigning values to this property.
- **top** An alternative for the name of the top-most window.
- **window** An alternative for the name of the current window.

Methods

- **alert(*message*)** Displays *message* in a dialog box.
- **close()** Closes the window.
- **confirm(*message*)** Displays *message* in a dialog box with OK and CANCEL buttons. Returns true or false based on the button clicked by the user.
- **open(*url*,*name*,*features*)** Opens *url* in a window named *name*. If *name* doesn't exist, a new window is created with that name. *features* is an optional string argument containing a list of features for the new window. The feature list contains any of the following name-value pairs separated by commas and without additional spaces:

toolbar=[yes,no,1,0]	Indicates if the window should have a toolbar
location=[yes,no,1,0]	Indicates if the window should have a location field
directories=[yes,no,1,0]	Indicates if the window should have directory buttons
status=[yes,no,1,0]	Indicates if the window should have a status bar
menubar=[yes,no,1,0]	Indicates if the window should have menus
scrollbars=[yes,no,1,0]	Indicates if the window should have scroll bars
resizable=[yes,no,1,0]	Indicates if the window should be resizable
width= <i>pixels</i>	Indicates the width of the window in pixels
height= <i>pixels</i>	Indicates the height of the window in pixels

- **prompt(*message*,*response*)** Displays *message* in a dialog box with a text entry field with the default value of *response*. The user's response in the text entry field is returned as a string.
- **setTimeout(*expression*,*time*)** Evaluates *expression* after *time* where *time* is a value in milliseconds. The time out can be named with the structure

name = setTimeout(*expression*,*time*)

- **clearTimeout(*name*)** Cancels the time out with the name *name*.

Event Handlers

- **onLoad** Specifies JavaScript code to execute when the window or frame finishes loading.
- **onUnload** Specifies JavaScript code to execute when the document in the window or frame is exited.

Independent Functions, Operators, Variables, and Literals

Independent Functions

- **escape(character)** Returns a string containing the ASCII encoding of *character* in the form %xx where xx is the numeric encoding of the character.
- **eval(expression)** Returns the result of evaluating *expression* where *expression* is an arithmetic expression.
- **isNaN(value)** Evaluates value to see if it is NaN. Returns a boolean value. This function is only available on UNIX platforms where certain functions return NaN if their argument is not a number.
- **parseFloat(string)** Converts *string* to a floating point number and returns the value. It continues to convert until it hits a non-numeric character and then returns the result. If the first character cannot be converted to a number the function returns "NaN" (zero on Windows platforms).
- **parseInt(string,base)** Converts *string* to an integer of base *base* and returns the value. It continues to convert until it hits a non-numeric character and then returns the result. If the first character cannot be converted to a number, the function returns "NaN" (zero on Windows platforms).
- **unescape(string)** Returns a character based on the ASCII encoding contained in *string*. The ASCII encoding should take the form "%integer" or "hexadecimalValue".

Operators

- **Assignment** Operators Table B.1. Assignment operators in JavaScript

Table B.1. Assignment operators.

Operator	Description

=	Assigns value of right operand to the left operand
+=	Adds the left and right operands and assigns the result to the left operand
-=	Subtracts the right operand from the left operand and assigns the result to the left operand
*=	Multiplies the two operands and assigns the result to the left operand
/=	Divides the left operand by the right operand and assigns the value to the left operand
%=	Divides the left operand by the right operand and assigns the remainder to the left operand

- **Arithmetic Operators** Table B.2. Arithmetic operators in JavaScript

Table B.2. Arithmetic operators.

<i>Operator</i>	<i>Description</i>
+	Adds the left and right operands
-	Subtracts the right operand from the left operand
*	Multiplies the two operands
/	Divides the left operand by the right operand
%	Divides the left operand by the right operand and evaluates to the remainder
++	Increments the operand by one (can be used before or after the operand)
--	Decreases the operand by one (can be used before or after the operand)
-	Changes the sign of the operand

- **Bitwise Operators** Bitwise operators deal with their operands as binary numbers but return JavaScript numerical value (see Table B.3).

Table B.3. Bitwise Operators in JavaScript.

<i>Operator</i>	<i>Description</i>
AND (or &)	Converts operands to integers with 32 bits, pairs the corresponding bits and returns one for each pair of ones. Returns zero for any other combination
OR (or)	Converts operands to integers with 32 bits, pairs the corresponding bits and returns one for each pair where one of the two bits is one. Returns zero if both bits are zero
XOR (or ^)	Converts operands to integer with 32 bits, pairs the corresponding bits and returns one for each pair where only one bit is one. Returns zero for any other combination.
<<	Converts the left operand to an integer with 32 bits and shifts bits to the left the number of bits indicated by the right operand—bits shifted off to the left are discarded and zeros are shifted in from the right
>>>	Converts the left operand to an integer with 32 bits and shifts bits to the right the number of bits indicated by the right operand—bits shifted off to the right are discarded and zeros are shifted in from the left
>>	Converts the left operand to an integer with 32 bits and shifts bits to the right the number of bits indicated by the right operand—bits shifted off to the right are discarded and copies of the leftmost bit are shifted in from the left

- **Logical Operators** Table B.4. Logical operators in JavaScript

Table B.4. Logical operators.

<i>Operator</i>	<i>Description</i>

&&	Logical "and"—returns true when both operands are true, otherwise it returns false
	Logical "or"—returns true if either operand is true. It only returns false when both operands are false
!	Logical "not"—returns true if the operand is false and false if the operand is true. This is a unary operator and precedes the operand.

- **Logical Operators** Table B.5. Comparison Operators in JavaScript

Table B.5. Logical (comparison) operators.

Operat or	Description
==	Returns true if the operands are equal
!=	Returns true if the operands are not equal
>	Returns true if the left operand is greater than the right operand
<	Returns true if the left operand is less than the right operand
>=	Returns true if the left operand is greater than or equal to the right operand
<=	Returns true if the left operand is less than or equal to the right operand

- **Conditional Operators** Conditional expressions take one form:

(condition) ? val1 : val2

If *condition* is true, the expression evaluates to *val1*, otherwise it evaluates to *val2*.

- **String Operators** The concatenation operators (+) is one of two string operators. It evaluates to a string combining the left and right operands. The concatenation assignment operator (+=) is also available.
- **Operator Precedence** JavaScript applies the rules of operator precedence as follows (from lowest to highest precedence):

comma (,)

assignment operators (= += -= *= /= %=)

conditional (? :)

logical or (||)

logical and (&&)

bitwise or (|)

bitwise xor (^)

bitwise and (&)

equality (== !=)

relational (< <= > >=)

shift (<< >> >>>)

addition/subtraction (+ -)

multiply/divide/modulus (* / %)

negation/increment (! - ++ --)

call, member () []