

Dynamic Hypertext Markup Language (DHTML)

Dieses Script entstand anlässlich einer Unterrichtseinheit für die Initiative „Schulen ans Netz“. Es verfolgt keinerlei kommerzielle Absichten und ist für den privaten, nicht-kommerziellen Gebrauch frei verfügbar und darf beliebig kopiert und weitergegeben werden.

Die Anwendung von DHTML sollte sparsam erfolgen. Der Anwender überprüfe seine Seiten doch stets mit einem 28.000er Modem - viele (wahrscheinlich die meisten) seiner Besucher sehen sie so.

Es sollten immer alternative Seiten ohne Frames, DHTML, JavaScript und Java existieren. Zum einen benutzen immer noch gut 5 % der Anwender den mit Windows NT mitgelieferten Internet Explorer 3.0 bzw 3.01 (keine Frames !) und eine weitere Schar von Usern hatte keine Lust auf die Megabyte-Parade von Microsoft und Netscape und surft mit dem Navigator 3.0 (Gold), d.h. kein DHTML, JavaScript nur bedingt. Um all diese User nicht zu verärgern, sollten die grundlegenden Informationen (wer bin ich?, wozu diese Web-Seite?, ...) auf einer Seite bereitgestellt werden, die wirklich alle lesen können.

Bei der Erstellung des Scripts habe ich mich sehr von Bernd Buss leiten lassen, dem ich auf diesem Weg meinen Dank aussprechen möchte.

(Kurz-) Gebet

(vor der Erstellung von Web-Seiten zu Sprechen)#

Windows User

Das du bist im Rechner, geheiligt seien deine Fenster.

Dein Crash komme. Dein Wille geschehe, wie in 95 so in 98.

Unser tägliches Update gib uns heute.

Und vergib uns unsere Linuxpartition,
wie auch wir vergeben den Treiberbugs.

Und führe uns nicht in den Bluescreen,
sondern erlöse uns von DLL-Versionskonflikten.

Denn dein ist das RAM und die Festplatte und die CPU-Zeit.

In Ewigkeit.

Alt-F4.

Inhaltsverzeichnis

Was ist DHTML?	4
Die Komponenten von DHTML	4
Vorteile und Nachteile von DHTML	5
Vorteile und Möglichkeiten	5
Nachteile und Einschränkungen	6
Cascading Style Sheets	6
Das Einbinden von Style Sheets	7
Style-Sheet-Dateien und Ausgabemedium	8
Stil-Klassen	9
Die Gruppierungsmethode	9
Die Zuordnung von Klassen	10
Das Document Object Model	11
Das Modell von Microsoft	11
Das Modell von Netscape	12
CrossBrowser DHTML	13
Der DIV-Container	16
Positionierung	16
Bewegende Momente	18
Von Koordinaten und Ebenen	18
Special Effects	21
Pop-Ups und Menus	21
Animierte Ebenen	23
Nichtlineare Bewegungen	25
Hardware + Layout	28
Wer surft denn da?	28
Plugins	31
Bildschirm	34
Troubleshooting	36
Links ins WWW	38

Was ist DHTML?

HTML ermöglicht es, Informationen in in sich abgeschlossene Häppchen zu zerteilen und über diese Informationseinheiten eine Verweisstruktur zu legen. Der Anwender ist nicht mehr gezwungen, wie bei anderen am Computer erstellten Texten, eine vorgegebene Lesereihenfolge einzuhalten. Mit Querverweisen/Hyperlinks kann er das Angebot durchstöbern. Auch deshalb werden inzwischen viele Hilfedateien und Programmdokumentationen in HTML geschrieben.

Die meisten Anwender wollen jedoch mehr. Immerhin haben sie viel Geld investiert, um das Internet zu durchstöbern, und sie erwarten nun etwas mehr als die elektronische Wiedergabe einer Buchseite. Diese Erwartungshaltung führt direkt zu Dynamic HTML und einer Seite, die veränderbar ist, nachdem der Server die Seite an den Browser gesendet hat.

DHTML ist keine klassische HTML-Erweiterung wie HTML 4.0 in Gestalt neuer TAG's und auch keine neue Sprache. DHTML ist eine Kombination aus mehreren Techniken, unter anderem HTML, Cascading Style Sheets, Scripting und Objektorientierter Programmierung, die zusammen eingesetzt werden, um Webseiten zu gestalten.

Dynamic HTML erlaubt es dem Entwickler von Webseiten, die Darstellung einer Seite zu ändern, indem er ein Style-Sheet anpaßt, und indem er auf die Eigenschaften, Methoden und Ereignisse für die HTML-Elemente zugreift, um diese Elemente dynamisch oder statisch zu positionieren oder ihr Erscheinungsbild zu ändern.

Die Komponenten von DHTML

Nach der Meinung von Netscape und Microsoft soll DHTML neben seinen dynamischen Aspekten auch die Webseiten-Präsentation und die statische Positionierung beinhalten.

Die wichtigsten Komponenten von DHTML sind:

- Cascading Style Sheets 1/2 (CSS) und JavaScript Accessible Style Sheets (JASS) zur Webseiten-Präsentation
- Das Bereitstellen von HTML-Elementen für das Scripting
- Das dynamische Ändern des Erscheinungsbildes von HTML-Elementen
- Das Verbergen, Anzeigen, Verschieben und Zuschneiden von HTML-Elementen

- Das Auffangen von Ereignissen für traditionelle Elemente
- Multimedia-Erweiterungen wie Übergangseffekte, Filter und Dynamic Fonts

Einige dieser Technologien haben beide Browser gemeinsam, andere sind proprietäre Eigenschaften von Microsoft und Netscape. Als CrossBrowser-DHTML wird die Anpassung an die Eigenschaften beider 4er-Browser bezeichnet. Der Begriff wurde von Kevin Lynch geprägt.

Folgende DHTML-Konzepte unterstützen beide Browser gemeinsam, obwohl sich ihre spezifische Unterstützung im Detail unterscheidet:

- Cascading Style Sheets
- Das Document Object Model
- JavaScript

Folgende Features sind browserspezifisch und arbeiten nicht mit dem jeweils anderen Browser zusammen:

- <layer>-Tag
- JavaScript Accessible Style Sheets
- Bitstream Fonts
- Direct Animation Controls
- Data Binding
- VBScript
- OpenType Fonts

Vorteile und Nachteile von DHTML

DHTML bringt einige neue Features zur Webseitengestaltung. Diese intensive Anwendung von DHTML bringt jedoch nicht nur Vorteile mit sich, es existieren auch einige Einschränkungen.

Vorteile und Möglichkeiten

- Die dynamische Änderung des Inhaltes nach dem Laden der Seite
- Pixelgenaue Plazierung von Inhalten ohne blinde Tabellen oder Gifs
- Einbindung multimedialer Inhalte wie Shockwave in Container
- Wechselnde Inhalte ohne Nachladen der Seite

- Die Stiländerung der Seite durch den Benutzer
- Die dynamische Änderung von Textattributen
- Pop-Ups für Links mit Erklärungen und Hinweisen

Nachteile und Einschränkungen

- Die Bildschirmgröße des Users muß bekannt und alternative Seiten sollten vorhanden sein.
- Es existiert das Problem der nachträglichen Veränderung der Fenstergröße durch den Benutzer. Dazu weiß ich zur Zeit leider auch nur eine Lösung, die mit Netscape funktioniert.
- Die beiden Browser haben eine unterschiedliche Fenstergröße im Hinblick auf die genaue Inhaltspositionierung mit den Style-Attributen 'left' und 'top'.
- Es sollte schon eine Bildschirmauflösung von mind. 800x600 beim User vorhanden sein und das Verwenden der 4er-Versionen der Browser ist obligatorisch.
- Das Ausdrucken von Inhalten in versteckten Containern ist nicht möglich.
- Ein absolut sauberer Code für beide Browser ist notwendig, alle Tags müssen abgeschlossen werden. Der kleinste Fehler führt zu katastrophalen Ergebnissen.

Trotzdem sollten Sie sich nicht abschrecken lassen DHTML einzusetzen. Es gibt für alle Probleme akzeptable Lösungen.

Cascading Style Sheets

Style Sheets werden direkt in die Seite eingebettet oder aus einer externen CSS-Datei geladen. Sie ermöglichen es, das Erscheinungsbild aller Elemente eines bestimmten Typs, einer Gruppe möglicherweise unterschiedlicher Elemente oder eines bestimmten Elementes anzupassen.

Ein einfaches Beispiel:

```
<STYLE type="text/css">
<! -- Verstecken vor älteren Browsern
H1 {color: red; font-family: Arial; margin: 10px}
// -->
</STYLE>
```

Das Style Sheet setzt den Stil für H1-Überschriften auf die Farbe Rot, die Schriftart Arial und definiert einen Rand von 10 Pixeln.

Sowohl Netscape als auch Microsoft haben den CSS1-Standard übernommen, obwohl keiner der beiden ihn so vollständig implementiert, wie er in der Level-1-Spezifikation des W3C definiert ist, was auch hier wieder zu unterschiedlichen Darstellungsweisen führen kann. Microsoft hat hier mehr getan als Netscape.

Das Einbinden von Style Sheets

Mit Hilfe der Meta-Angabe kann einem Browser ebenfalls mitgeteilt werden, daß diese Datei Style-Sheet-Definitionen enthält:

```
<HEAD><TITLE>CSS Inhalt</TITLE>
<META HTTP-EQUIV="CONTENT-STYLE-TYPE" CONTENT="text/css">
</HEAD>
```

Die Angabe wird von den Browsern der Hersteller Netscape und Microsoft nicht verlangt. Laut Vorgabe des W3-Konsortiums sollte sie aber gemacht werden, um die Kommunikation zwischen Server und Client (= Browser) zu erleichtern.

Wird eine andere Style-Sheet-Sprache als CSS verwendet, muß anstelle von „text/css“ der entsprechende MIME-Typ der Sprache angegeben werden.

Es gibt vier Techniken, mit denen Style Sheet Definitionen in eine Webseite aufgenommen werden können:

- Das Style Sheet wird in den Header-Bereich des Dokumentes eingebunden.
- Ein externes Style Sheet wird gelinkt.
- Ein externes Style Sheet wird importiert.
- Eine Inline-Definition eines Style Sheets wird direkt in ein HTML-Element übernommen.

Das obige Beispiel zeigt die erste Version mit einem Kommentar, um den Inhalt vor älteren Browsern zu verbergen.

Ein Beispiel für ein externes Style Sheet:

```
<HEAD><TITLE>CSS Inhalt</TITLE>
<LINK REL=STYLESHEET TYPE="text/css" HREF="style1.css"
TITLE="stylesheet1">
</HEAD>
```

Die ASCII-Datei style1.css, welche die Stildefinitionen enthält, hat dabei folgende Form:

```
<STYLE type="text/css">
BODY {font-family: ARIAL, sans-serif; font-style: normal;
background-color: #3b3b3b}
A {text-decoration: none}
P.ident {text-indent : 10px; text-align: left}
</STYLE>
```

Wenn Sie ein externes Style Sheet mit LINK REL=stylesheet einbinden und gleichzeitig META-Tags nutzen für eine automatische Weiterleitung auf eine andere Seite

```
<meta http-equiv="refresh" content="10; URL=url/start.htm">
```

darf der Link zum Style Sheet erst nach Meta-Refresh stehen!

Eine weitere Möglichkeit, eine externe Datei zu nutzen, ist das Importieren des Style Sheets in die Datei. Die Datei wird dabei automatisch in die Webseite eingefügt.

Der Code dazu hat folgende Form:

```
<STYLE type="text/css">
@import url (style1.css)
</STYLE>
```

Die letzte Technik ist die direkte Einbettung in ein HTML-Element. Die Stildefinition gilt dann auch nur für dieses eine Element:

```
<P style="color: red; font-style: italic">
Der Absatz enthält kursiven Text in roter Farbe
</P>
```

Stilattribute können Sie definieren für Schriften, Farbe und Hintergrund, Textausrichtung und Darstellung, Ränder, Rahmen und Abstände und als Klassifizierung für das Verhalten anderer Elemente wie Listen.

Style-Sheet-Dateien und Ausgabemedium

Das W3-Konsortium hat Möglichkeiten festgelegt, den unterschiedlichen Möglichkeiten verschiedener Ausgabegeräte (Bildschirm, Drucker, ...) Rechnung zu tragen, d.h. verschiedene Ausgabemedien können verschiedene Style-Sheet-Angaben nutzen.

Die Einbindung in die HTML-Datei erfolgt über den Meta-Tag:

```
<HTML>

<HEAD>

<TITLE>CSS</TITLE>

<LINK REL=STYLESHEET MEDIA="screen" href="url/css1.css">

<LINK REL=STYLESHEET MEDIA="print" href="url/css2.css">

</HEAD>

<BODY>

</BODY>

</HTML>
```

Als Angaben bei „Media“ ist bis jetzt folgendes vorgesehen:

- MEDIA="screen" für die Bildschirmpräsentation,;
- MEDIA="print" für die Drucker-Ausgabe;
- MEDIA="aural" für die SWprachausgabe via Lautsprecher;
- MEDIA="projection" für die Ausgabe über Dia- / Overheadprojektor;
- MEDIA="braille" für die Ausgabe über Braille-Medien;
- MEDIA="tv" für die Ausgabe über Fernseher;
- MEDIA="handheld" für die Ausgabe über Palmtops, Handys, usw.;
- MEDIA="all" für Style-Sheets, die in allen Medientypen gelten.

Sowohl Microsoft wie auch Netscape wollen die Angaben in Ihren nächsten Browsern implementieren, z. Zt. finden sie jedoch noch keine Beachtung

Stil-Klassen

Es gibt zwei Möglichkeiten der Organisation von Formatvorlagen.

Die erste wird als Gruppierung bezeichnet, es werden logische Formatgruppen gebildet. Wenn Sie ein Format verändern wollen, können Sie Klassen verwenden. Diese Methode ermöglicht Ihnen, demselben HTML-Tag dynamisch per Scripting verschiedene Formate zuzuweisen.

Die Gruppierungsmethode

Diese Methode weist demselben Tag dieselben Formatattribute zu:

```
H1, H2, H3 {font-family: arial; font.size: 14pt; color: red}
```

Sie können auch Attribute in einem Format zu Gruppen zusammenfassen, indem Sie eine Reihe von Attributen in eine Kategorie eingeben.

Anstatt folgendes zu schreiben, wird das zweite Beispiel benutzt:

```
BODY {font-family: arial, sans-serif; font-size: 12pt; font-weight: bold; font-style: italic}

BODY {font: bold italic 12pt arial, sans-serif}
```

Zu beachten dabei ist:

- Variablen werden nur durch ein Leerzeichen getrennt.
- Es kommt auf die Reihenfolge der Attribute an. Schriftschnitt und -stil müssen vor anderen Schriftattributen stehen; die Schriftgröße kommt vor einer eventuellen Angabe der Linienhöhe.
- Danach können zusätzliche Informationen angegeben werden.

Die Zuordnung von Klassen

Wenn Sie Elemente innerhalb einer Webseite dynamisch verändern wollen, müssen Sie Stilklassen definieren. Mit dem Klassennamen kann ein Element per Scripting eindeutig angesprochen und verändert werden. Die Zuweisung von Klassen geschieht, indem Sie eine Namenserverweiterung an einen beliebigen HTML-Tag anhängen:

```
P.links {font-family: arial; font-size: 10pt; font-style: normal; text-align: left}

P.rechts {font-family: arial; font-size: 10pt; font-style: normal; text-align: right}
```

Es werden zwei neue <P>-Formate erzeugt mit jeweils anderer Textausrichtung. Diese werden im Dokument als <P class="links"> und <P class="rechts"> verwendet.

Setzen Sie jetzt beide Gruppierungsarten ein, erhalten Sie folgenden Code:

```
P {font: arial 10pt normal}

P.links {text-align: left}

P.rechts {text-align: right}
```

Im Body-Teil steht dann folgender Code:

```
<BODY>

<P class="links">Dieser Text erscheint linksbündig in Arial
10pt hoch</P>
```

```
<P class="rechts">Dieser Text erscheint rechtsbündig in  
Arial 10pt hoch</P>  
</BODY>
```

Das Document Object Model

Ziel von DHTML ist es, alle Elemente einer Webseite dynamisch veränderbar zu machen. Dazu muß mit einer Scriptsprache wie JavaScript gezielt auf bestimmte HTML-Elemente zugegriffen werden können.

Das DOCUMENT OBJECT MODEL versucht alle relevanten Bestandteile einer angezeigten Webseite als eine Objekthierarchie abzubilden. Diese Hierarchie sollte sich dann in objektorientierten Programmiersprachen wie JavaScript wiederfinden.

Das W3C hat einen Vorschlag zu einem Modell veröffentlicht. In der aktuellen HTML-Version 4.0 ist festgelegt, welche HTML-Befehle welche Event-Handler wie 'mouseOver' zum Steuern von Ereignissen haben können. Das DOM ist zwar ein Konzept beider Browser, aber wie immer sind die Ansätze von Netscape und Microsoft verschieden, eigentlich sogar entgegengesetzt, wobei Microsoft das DOM weitgehend umgesetzt und Netscape für die 5er-Version Besserung versprochen hat.

Um Scripts zu schreiben, die mit beiden Browsern funktionieren, ist es z.Z. noch notwendig jedes Ereignis als einzelne Funktion zu behandeln. Man muß sich also an die strenge Netscape-Objekthierarchie halten, da diese auch vom Explorer unterstützt wird.

Das Modell von Microsoft

Vom Internet-Explorer wird der Scriptzugang zu allen HTML-Seitenelementen unterstützt, einschließlich der Style Sheets. Seitenelemente werden als Objekte enthalten in einer document.all-Sammlung erfaßt und können durch Index, Name oder ID angesprochen werden.

Beispiel: den Namen aller Tags einer Seite schreiben

```
for (i=0;i<document.all.length;i++)  
{document.write(document.all[i].tagName + "\n");}
```

Der IE benutzt dazu ein 'Event bubbling model':

Aktionen, die von einem hierarchisch niedrigeren Objekt kommen, werden an die höherwertigen Elemente weitergereicht (Luftblasen-Prinzip). Jedes Seitenelement kann Ereignisse erzeugen und ausführen. Jedes simple HTML-Tag wie <H1> oder

<BLOCKQUOTE> kann so mouseOver, mouseOut oder onClick-Ereignisse ausführen.

Ein Trick, um ähnliches mit dem Navigator zu erreichen, ist, das Element in ein leeres ANCHOR-Tag einzuschließen. Ein leeres ANCHOR-Tag verweist nicht auf eine URL oder NAME, wie der folgende Code darstellt:

```
<H1><A HREF="" onClick="eine_funktion();return false">
Die Überschrift</A></H1>
```

Um die Farbe und die Textdekoration für den Anker-Link zu entfernen, wird dem ANCHOR-Tag ein Style Sheet zugewiesen: A {text-decoration: none; font-color: wie Text}. Für den Ereignis-Handler von 'onClick' muß der Wert 'false' zurückgegeben werden, sonst wird das Ereignis von dem Anker-Element verarbeitet. Weil das Dokument auf eine leere URL verweist, wird dann das Verzeichnis angezeigt, indem das Dokument enthalten ist.

Das Modell von Netscape

Von Netscape wird der Scriptzugang nur zu einer spezifischen Anzahl von Seitenelementen unterstützt, z.Bsp. den Layern einer Seite. Layer im Navigator beinhalten einen Seitenabschnitt/Container, begrenzt vom LAYER-Tag und die Positionierung durch CSS-Attribute.

Der Navigator benutzt das sogenannte 'Capturing model', das Gegenteil des 'Event bubbling models'. Dies besagt, daß die Aktionen von Mutter-Objekten zu Tochter-Objekten durchgereicht werden. Die Objekte, mit denen der User interagieren kann, sind streng limitiert. Es sind:

Formulare, PlugIns, Frames, Anchors, Links, Layer, Applets und Bilder.

Statisch hingegen bleiben die restlichen Elemente. Die speziellen Elemente können durch Name, ID oder Index angesprochen werden.

Beispiel: den Namen aller Layer einer Seite schreiben

```
for (i=0;i<document.layers.length;i++)
{document.write(document.layers[i].name + "\n");}
```

Bei beiden Browsern erscheint jede Änderung der Objekteigenschaften augenblicklich auf der Seite.

CrossBrowser DHTML

Es gibt einige Tricks, mit deren Hilfe Sie browserübergreifende Seiten erstellen können.

Sowohl Netscape als auch der IE unterstützen den DIV-Block und beide unterstützen die statische CSS-Positionierung für diese Elemente. Sie sollten also besser den DIV-Tag für Container benutzen statt den LAYER-Tag, zumal auch Netscape die Verwendung von Layern inzwischen nicht mehr empfiehlt.

Der IE beinhaltet mehr Ereignisse für HTML-Elemente als der Navigator, aber durch die Verwendung des schon beschriebenen leeren ANCHOR-Tags, können die Möglichkeiten beider Browser angeglichen werden.

Darüberhinaus ist noch das Anlegen separater Codeblöcke möglich oder Prüfungen innerhalb eines Codeblocks.

Um Objekte in beiden Browsern ansprechen zu können, wird eine Referenz abgefragt wenn die Seite geladen wird, um zu erfahren welcher Browser vom User genutzt wird. Dazu wird LAYER von Netscape benutzt, da der IE diesen Tag nicht kennt.

```
if (document.layers){ns=1; ie=0;}
else {ns=0; ie=1;}
```

Es wird überprüft, ob der benutzte Browser das LAYER-Objekt kennt. Wenn ja, wird der Variablen ns für den Navigator das Flag 1 für 'true' und der Variablen ie das Flag 0 für 'false' zugewiesen. Wird das Objekt nicht erkannt, ist die Zuweisung umgekehrt. Um die ermittelten Werte weiter verwenden zu können, wird eine Funktion zur Initialisierung des geladenen Dokumentes benötigt:

```
function init()
{
    if (document.layers)
    {
        ns=1;
        ie=0;
        ebene=document.nameEbene;
    }
    else
    {
```

```

        ns=0;

        ie=1;

        ebene=nameEbene.style;
    }
}

```

Diese Funktion wird im BODY-Tag mit 'onLoad' eingebunden: <BODY onLoad="init()">

Auch die Verwendung folgenden Crossbrowser-APIs von Macromedia ist möglich und sinnvoll:

```

if (navigator.appName=="Netscape")
{layerRef="document.layers";styleRef="";}
else {layerRef="document.all";styleRef=".style";}

```

Im Prinzip geht es immer nur um die Abfrage des verwendeten Browsers und um die Umsetzung des Ergebnisses in wenn Netscape, dann document.layers nutzen; wenn IE, dann document.all verwenden, dh. die Berücksichtigung der verschiedenen DOM-Ansätze beider Browser.

Wie Sie sicher schon inzwischen gemerkt haben, werden Sie nicht daran vorbeikommen, sich mit JavaScript zu beschäftigen, wenn Sie DHTML einsetzen wollen.

Um das besser zu verdeutlichen, hier jetzt ein kleines Beispiel.

Die unsichtbare Ebene, die Sie mit einem Mausklick anzeigen und verstecken können enthält den Code für dieses Beispiel:

Im HEADER stehen die Browserabfrage, die Abfrage nach dem LAYER-Objekt und die beiden Funktionen 'Ebene anzeigen' und 'Ebene verbergen':

```

<SCRIPT language="javascript">
<!-- auskommentieren für Browser ohne JavaScript
if (((navigator.appName == "Netscape") &&
(parseInt(navigator.appVersion.substring(0,1)) >=3)) ||
((navigator.appName == "Microsoft Internet Explorer")&&
(parseInt(navigator.appVersion.substring(0,1)) >=4)))
{var version=true}
/* Browserabfrage: Layer-Object bekannt? */
if (document.layers) {ns = 1; ie = 0;} else {ns = 0; ie = 1;}

```

```

/* Funktion Ebene BEISPIEL anzeigen */
function show(name)
{
if (ns)
{document.layers['' + name].visibility = "show";}
// Navigator
else
{document.all['' + name].style.visibility = "visible";}
/ Internet Explorer
}
/* Funktion Ebene BEISPIEL verbergen */
function hide(name)
{
if (ns)
{document.layers['' + name].visibility = "hide";}
else
{document.all['' + name].style.visibility = "hidden";}
}
// -->
</SCRIPT>

```

Im BODY steht der folgende Code für die Ebene BEISPIEL als DIV-Container. Die Besonderheiten zur Einbindung der Container werden im nächsten Kapitel erklärt.

```

<DIV ID="Beispiel" style="position: relative; width: 560;
height: 800; visibility: hide; visibility: hidden">
<TABLE WIDTH=560 HEIGHT=800 BORDER=1 BGCOLOR="#FEFFED"
CELLPADDING=5 CELLSPACING=0><TR><TD ALIGN=left Valign=top>
Hier kommt der Inhalt der Ebene, wobei die Tabelle natürlich
nicht nur auf eine Zeile und eine Spalte beschränkt sein muß.
</TD></TR></TABLE>
</DIV>

```

Für das Anzeigen und Verbergen der Ebene habe ich hier nur den ANCHOR-Tag mit JavaScript benutzt anstatt den leeren ANCHOR-Tag:

```

<A HREF="Javascript:show('Beispiel')"><EBENE ANZEIGEN></A>
<A HREF="Javascript:hide('Beispiel')"><EBENE VERBERGEN></A>

```

Der DIV-Container

Damit Scriptsprachen die einzelnen Container/Elemente unterscheiden können, benötigen diese eindeutige Bezeichnungen per ID, NAME oder INDEX. Im vorherigen Beispiel mit ID="Beispiel". Diese eindeutige Zuordnung war bisher nur mit Einschränkungen möglich. Dafür kann jetzt der DIV-Tag benutzt werden.

Der DIV-Tag faßt alle Inhalte zu einer einzigen Instanz zusammen, einem Container. Formatierungen, die als Attribute im DIV-Tag eingetragen sind, wirken sich auf alle darin enthaltenen Elemente aus (es sei denn, ein Element verfügt über ein eigenes Attribut gleichen Names ---> Cascading Style Sheets).

```
<DIV ID="Container" ALIGN=center VALIGN=top>
<H3>Inhalt</H3><BR>
<P ALIGN=right><IMG SRC="bild.gif"></P>
</DIV>
```

Im Container wird die H3-Überschrift zentriert und oben angezeigt, das Bild jedoch rechts, weil der P-Absatz ein eigenes ALIGN-Attribut hat.

Damit Netscape einen Container ganz ausfüllt (z.Bsp. Hintergrundfarbe) und nicht nur den verwendeten Teil (Textbereich), muß eine Tabelle in den Container geladen werden mit den gleichen WIDTH- und HEIGHT-Angaben wie beim Container.

Die Angabe eines 1x1px-Bildes beim DIV-Tag geht noch als Hintergrund, aber nicht mehr bei anderen Inhalten. Zumal dies ein äußerst unzureichender Trick ist.

Auch die Einbindung multimedialer Inhalte wie Shockwave verbunden mit pixelgenauer Positionierung ist so möglich.

Positionierung

Mit der Positionierung von HTML-Elementen mit CSS werden die Grenzen von HTML aufgehoben. Sie sind nicht mehr darauf angewiesen komplizierte, blinde Tabellen zu konstruieren, nur um ein Bild an einer bestimmten Stelle der Seite anzuzeigen.

Elemente erscheinen an der Stelle, an der Sie es wünschen, egal wo sie im HTML-Code stehen. Die Elemente können sich überlagern, und Sie können die Schichtreihenfolge bestimmen.

Die ist wahrscheinlich die wichtigste Eigenschaft von CSS.

Hier ein Beispiel, in dem drei farbige Ebenen mit dem folgenden Code erzeugt werden:

```

<DIV ID="Layer1" style="position: absolute; width: 100;
height: 100; top:600; left: 300; z-index:1; visibility: hide;
visibility: hidden">
<TABLE WIDTH=100 HEIGHT=100 BGCOLOR=red><TR><TD>
Leerzeichen</TD></TR></TABLE>
</DIV>

<DIV ID="Layer2" style="position: absolute; width: 100;
height: 100; top:300; left: 600; z-index:2; visibility: hide;
visibility: hidden">
<TABLE WIDTH=100 HEIGHT=100 BGCOLOR=yellow><TR><TD>
Leerzeichen</TD></TR></TABLE>
</DIV>

<DIV ID="Layer3" style="position: absolute; width: 100;
height: 100; top:50000; left: 500; z-index:3; visibility:
hide; visibility: hidden">
<TABLE WIDTH=100 HEIGHT=100 BGCOLOR=blue><TR><TD>
Leerzeichen</TD></TR></TABLE>
</DIV>

```

Es werden 3 Container Layer1/Layer2/Layer3 erzeugt, hier alle mit den gleichen Ausmaßen.

Die Style-Angabe ist bei allen 'absolute' gesetzt, dh. es müssen mit den Angaben 'left' und 'top' die Entfernung der Ebene vom linken Rand und vom oberen Rand des Browserfenster angegeben werden.

Alle 3 Container haben unterschiedliche Abstandswerte.

Allen Containern wird ein Z-Index zugeordnet, der die Reihenfolge auf dem Stapel definiert; der höchste Z-Index ist der oberste Container, hier die blaue Ebene mit 3. Da die blaue Ebene den höchsten Z-Index hat, überlagert sie die anderen Ebenen.

Die Ebenen sind beim Laden der Seite alle unsichtbar. Dies wird mit der Style-Eigenschaft 'visibility' festgelegt; hier mit hide für den NS und hidden für den IE. Damit das CrossBrowser-Modell funktioniert, müssen im DIV-Container zwei Visibility-Werte angegeben werden, für Netscape und Microsoft:

visibility: show/visible und visibility: hide/hidden.

Vier Punkte 1-4 dienen als Auslöser für das Anzeigen der Ebenen.

```
<A  
HREF="Javascript:show('Layer1');hide('Layer2');hide('Layer3')  
>
```

Punkt 1

```
<A  
HREF="Javascript:hide('Layer1');show('Layer2');hide('Layer3')  
>
```

Punkt 2

```
<A  
HREF="Javascript:hide('Layer1');hide('Layer2');show('Layer3')  
>
```

Punkt 3

Bei Punkt 4 werden alle Layer auf 'show' gesetzt:

```
<A  
HREF="Javascript:show('Layer1');show('Layer2');show('Layer3')  
>
```

Punkt 4

Der Unterschied zwischen absoluter und relativer Positionierung besteht darin, daß sich eine absolute Angabe auf den Fensterrand bezieht, eine relative Angabe dagegen auf ein voranstehendes Objekt. Deshalb wird bei der relativen Angabe der Raum gelassen zum Anzeigen des Containers.

Bewegende Momente

Von Koordinaten und Ebenen

Um eine Ebene zu verschieben wird sie 'absolut' positioniert mit den Koordinaten 'top:y' und 'left:x'. Diese Koordinaten werden nachträglich durch eine JavaScript-Funktion verändert zu 'top:(y + n)' und 'left:(x + n)'.

Es wird wieder ein Container erzeugt, diesmal mit einem Bild als Inhalt, der absolut positioniert wird mit den Koordinaten 'left:x=300' und 'top:y=610'. Dieser Container wird jetzt mit JavaScript-Funktionen manipuliert.

```
<SCRIPT language="javascript">
```

```

<!-- auskommentieren für Browser ohne JavaScript

/* Browserabfrage: Layer-Object bekannt? */
if (document.layers)
{
ns = 1; ie = 0;
}
else
{
ns = 0; ie = 1;
}

/* CrossBrowser */
function init()
{
if (ns) ebene=document.Ebene;
if (ie) ebene=Ebene.style;
}

/* Funktion verschieben */
function move(x,y)
{
ebene.top=x;
ebene.left=y;
}

```

Der Funktion 'move' werden die x/y-Koordinaten anhand von Parametern übergeben.

```
<A HREF="Javascript: move(610,30)">Pfeil links</A>
```

Die ursprünglichen Koordinaten 300/610 werden durch die neuen Koordinaten 30/610 ersetzt. Da 'left' mit x=30 jetzt kleiner ist als vorher mit x=300 wird die Ebene nach links verschoben. Beim rechten Pfeil ist es umgekehrt: von x=30 zu x=300.

Beim Scrollen wird die Bewegung in einzelne Schritte unterteilt, so daß sich die Ebene in 5-Bildpunkt-Schritten ans Ziel schiebt. Je kleiner die Schritte, desto flüssiger wird die Bewegung; aber auch langsamer.

Die Anweisung `Ebene.left +=5`; verschiebt die Ebene um 5 Pixel, indem zu aktuellen x-Koordinate 5 Pixel addiert werden und zwar nach Rechts. Mit `-=5` erfolgt die Bewegung nach Links.

Diese Bewegung wird kontinuierlich bis zum Ziel durchgeführt. Dazu wird die `setTimeout`-Methode benutzt. Um zu überprüfen, ob die Ebene am Ziel ist wird die `if`-Abfrage verwendet. `setTimeout` ruft die Funktion alle x-Millisekunden erneut auf.

Um Fehlermeldungen bei `setTimeout` beim IE zu vermeiden, sollten Sie die angegebene Syntax verwenden. Da der IE bei Positionsangaben `px` erwartet (10px anstatt 10) wird die Zeile `Ebene.left = ebene.x` eingefügt. Da der Navigator Zahlen erwartet muß der übergebende String mit `parseInt()` in einen Integer-Wert umgewandelt werden. Übernehmen Sie diesen Teil einfach als notwendig in jede Funktion.

Es werden jetzt 2 Funktionen definiert: einmal Scrollen-links und einmal Scrollen-rechts:

```
/* Funktion scrollen links */
function scrollenl (ziel,laufweite,delay) /* delay =
Verzögerung */
{
ebene.x = parseInt(ebene.left)
if (ebene.x > ziel)
{
ebene.x -=laufweite;
ebene.left=ebene.x
setTimeout('scrollenl
('+ziel+', '+laufweite+', '+delay+)',delay);
}
}

/* Funktion scrollen rechts */
function scrollenl (ziel,laufweite,delay)
{
ebene.x = parseInt(ebene.left)
if (ebene.x < ziel)
{
```

```

ebene.x +=laufweite;

ebene.left=ebene.x

setTimeout('scrollenl
('+ziel+', '+laufweite+', '+delay+')',delay);
}
}

```

Die Funktion init() wird mit <BODY onLoad="init()"> geladen.

Zur Auslösung der Funktionen wird wieder der ANCHOR-Tag mit Javascript benutzt:

```

<A HREF="Javascript:scrollenl(30,5,50)"><Pfeil links></A>

<A HREF="Javascript:scrollenr(300,5,50)"><Pfeil rechts></A>

```

Special Effects

Pop-Ups und Menus

Die Funktionen Anzeigen, Verbergen und Verschieben von Ebenen können sinnvoll eingesetzt werden um Pop-Up-Fenster als Hilfe oder Link-Erklärungen zu erstellen. Ebenso möglich ist das Erzeugen von aufklappbaren Menus.

Zum Erzeugen von Pop-Up-Fenstern werden einfach wieder die Funktionen function show() und function hide() benutzt. Als Auslöser werden werden die sicher schon von animierten Buttons bekannten mouseOver und mouseOut verwendet.

Eine feine Sache sind sicherlich Menus, die keinen Platz auf der Seite beanspruchen wenn Sie nicht benutzt werden. Einige kennen auch sicherlich die Netscape-Seite mit den hereinfliegenden Layern oder das Netcaster-Menü, das rechts nur mit einer Lasche angezeigt wird bei Nichtbenutzung: alles DHTML und keine Hexerei.

Ein Menufeld links, vom dem beim Start der Seite nur ein schmaler Streifen mit den Steuerungspfeilen zu sehen ist, erstellen Sie folgendermaßen (die Maßangaben sind nur Beispiele zu dieser Seite):

Sie erzeugen einen DIV-Container mit WIDTH:170 und HEIGHT:500 und der ID="MENU", positionieren ihn absolut und setzten VISIBILITY auf Anzeigen. In den Container laden Sie natürlich wieder die entsprechende Tabelle mit BORDER=1.

Die Positionsangaben der Ebene sind 'left: -130' und 'top: 200'. So ist beim Laden der Seite nur ein 40 Pixel breiter Streifen zu sehen, in den Sie Steuerungszeichen wie Pfeile oder Text einfügen können.

Mit der schon bekannten Funktion `function move()` und den Steuerungszeichen verschieben Sie die Ebene:

```
/*Menu-Ebene verschieben rechts */
function mover()
{
  if(ns)
  {
    document.MENU.left=-20;
  }
  else
  {
    MENU.style.left=-20;
  }
}

/*Menu-Ebene verschieben links */
function movel()
{
  if(ns)
  {
    document.MENU.left=-130;
  }
  else
  {
    MENU.style.left=-130;
  }
}
```

In die Tabelle des Containers, der ja beim Ausklappen mit 150 Pixel sichtbar ist, setzen Sie dann Ihre Text- oder Imagelinks.

So können Menus mit mehreren Untermenüs erzeugt werden. Dabei sollten Sie jedoch des besseren Handlings halber noch zwei Variablen hinzufügen: eine, welche die zuletzt aufgeklappte Ebene speichert, und eine, die speichert, wenn ein ausgefahrenes Untermenü beim Klicken in den Rest des Fensters wieder geschlossen wurde.

Ebenfalls benutzen können Sie die schon bekannten Funktionen `function show(Ebene)` und `function hide(Ebene)`.

Animierte Ebenen

Hier jetzt eine Funktion kennen, die Ebenen auf einer geraden Linie von einem Startpunkt zu einem Endpunkt bewegt; im Prinzip nichts anderes als die Funktion `Scrollen` mit einigen Ergänzungen: die Funktion kann gleichzeitig verschiedene Ebenen bedienen, die Geschwindigkeit und der Startzeitpunkt jeder Ebene kann angegeben werden.

Wenn die Ebene(n) beim Start nicht zu sehen sein soll, setzen Sie den Y-Wert auf '-100'. Damit ist sie außerhalb des sichtbaren Bereiches.

Im Scriptteil müssen Sie lediglich die Variable `var delay=x;` definieren, die die Verzögerung und Geschwindigkeit der Bewegung bestimmt.

Die Funktion `function blende` bekommt alle Werte als Parameter übergeben:

```
function blende (was, startx, starty, endx, endy, steps,
wait)
```

- WAS enthält den Namen der Ebene, die bewegt werden soll
- STARTX enthält den Startwert für die X-Koordinate der Ebene
- STARTY enthält den Wert für die Y-Koordinate
- ENDX gibt den Endwert für die X-Koordinate an
- ENDY gibt den Y-Endwert an
- STEPS gibt die Anzahl der Zwischenschritte an und steuert damit die Geschwindigkeit
- WAIT gibt an, wie lange mit dem Start der Bewegung gewartet werden soll

Als erstes wird getestet, ob der Parameter `wait` vorhanden und größer Null ist. Ist dies der Fall, wird die Funktion wieder mit `setTimeout()` und der Verzögerung `delay` aufgerufen. Dabei werden alle Parameter mit den vorhandenen Werten übergeben. Lediglich `wait` wird heruntergezählt mit -1 bis `wait` bei Null ist.

```
if (wait && wait>0)
setTimeout('blende ("'+was+'", '+startx+', '+starty+',
'+endx+', '+endy+', '+steps+', '+wait-1+')', delay);
```

Der folgende Block wird dann ausgeführt, wenn wait nicht angegeben oder inzwischen bei Null ist.

Unter Beachtung der verschiedenen Browser werden die Koordinaten der Ebene (was) neu gesetzt:

```
else
{
    if(document.layers)
    {
        document.layers[was].left=startx;
        document.layers[was].top=starty;
    }
    else
    {
        document.all[was].style.left=startx;
        document.all[was].style.top=starty;
    }
}
```

Jetzt werden die x/y-Koordinaten für den jeweils nächsten Schritt berechnet. Die Formel ist sehr einfach: $endx - startx$ ergibt den noch verbleibenden Abstand zum Endpunkt. Teilt man den Abstand durch die Anzahl der Zwischenschritte (steps), erhält man den Wert, um den die Ebene in einem Schritt weiterbewegt werden muß.

Dieser Wert wird jetzt mit += zu startx/starty addiert. Die Richtung ergibt sich daraus, ob der Endpunkt unter- oder oberhalb des Startpunktes ist.

Jetzt wird die Blendenfunktion mit setTimeout erneut aufgerufen bis die Variable steps bei Null angelangt ist:

```
startx+=(endx-startx)/steps;
starty+=(endy-starty)/steps;

if (steps>0)
setTimeout('blende(""+was+"", '+startx+', '+starty+',
'+endx+', '+endy+', '+ (steps-1) +')', delay);}}
```

Als letztes folgt die Funktion function init(), die die Ebenen der Reihe nach startet und die aktuelle Fenstergröße ermittelt - wichtig für den Endpunkt. Die Ermittlung der Fenstergröße erfolgt natürlich wieder browserspezifisch:

```

function init()
{
    if (navigator.appName=="Netscape")
    {
        var w=innerWidth;
        var h=innerHeight;
    }
    else
    {
        var w=document.body.offsetWidth;
        var h=document.body.offsetHeight;
    }
    var mw=w/2;
    var mh=h/2;
}

```

Die letzten beiden Variablen definieren den Mittelpunkt des Fensters. Jede der folgenden Zeilen bewegt eine Ebene. Im BODY-Teil wird wieder die die Funktion `init()` gestartet.

```

blende("Ebene1", 0, -100, h+10, 80, 20, 30);
blende("Ebene2", .....);

```

Jetzt noch 4 Funktionen, mit denen eine fensterfüllende Blende bewegt wird. Dazu wird ein Container mit den Angaben `WIDTH:101%` und `HEIGHT:101%` und der entsprechenden Tabelle erzeugt.

- Von oben nach unten - `blende("Ebene", w , 0, w, h+10, steps, delay)`
- Von unten nach oben - `blende("Ebene", w, h+10, w, 0, steps, delay)`
- Von links nach rechts - `blende("Ebene", 0, h, w+10, h, steps, delay)`
- Von rechts nach links - `blende("Ebene", w, h, -10, h, steps, delay)`

Nichtlineare Bewegungen

Um nichtlineare Bewegungen auszuführen, müssen die einzelnen Flugkoordinaten - der Pfad der Bewegung - in einem Feld abgespeichert und von einer Funktion verarbeitet werden. Für jede Flugbahn einer Ebene wird ein Array definiert, das die Flugkoordinaten enthält.

```
var Flug1=new Array (x1,y1, x2,y2, x3,y3, ... xn,yn);
var Flug2=new Array (x1,y1, x2,y2, x3,y3, ... xn,yn);
```

Die Flugdaten können willkürlich angeben oder aufgezeichnete Daten benutzt werden. Für die Datenaufzeichnung werden wir Ihnen am Schluß ein kleines Programm vorstellen.

Wie im vorherigen Kapitel wird wieder nur eine einzige Funktion function fliegen benötigt, der alle Werte als Parameter übergeben werden:

```
function fliegen (was, feld, step, dir, offsx, offsy, mode,
wait)
```

- WAS enthält den Namen der Ebene, die bewegt werden soll
- FELD ist das Array, indem die Flugdaten enthalten sind
- STEP ist der Arraypunkt, an dem die Bewegung sich gerade befindet. Durch andere Werte als Null wird der Startpunkt der Bewegung verschoben. Da es immer Koordinatenpaare sind, muß dieser Parameter immer durch 2 teilbar sein.
- DIR bestimmt die Richtung und Geschwindigkeit. Auch hier besteht eine Position wieder aus 2 Werten. Wenn Sie anstatt 2 4 oder 6 oder n angeben werden Bewegungspunkte übersprungen.
- OFFSX positioniert die gesamte Bewegung im Fenster anders. Zu jedem Punkt wird der Wert von 'offsetx' zur x-Koordinate hinzuaddiert.
- OFFSY das gleiche für die y-Koordinate
- MODE definiert die Bewegungsart wie bei FLASH:
 - 'Repeat' wiederholt die Bewegung immer wieder ab dem Anfangspunkt.
 - 'Loop' kehrt bei Erreichen der Endposition die Bewegung um.
 - Bei jeder anderen Angabe wird nur eine einmalige Bewegung durchgeführt.
- WAIT gibt an, wie lange mit dem Start der Bewegung gewartet werden soll.

Als erstes wird wieder die Verzögerung behandelt. Ist der Parameter angegeben und noch nicht bei Null wird er wieder heruntergezählt.

```
if (wait && wait>0)
```

```

setTimeout('fliegen ("'+was+"", "'+flug+", '+step+',
'+dir+', '+offsx+', '+offsy+', "'+mode+", '+wait-1)+'',
delay);

```

Die Variable 'step' gibt die momentane Stelle im Flugfeld an, repräsentiert durch die x/y-Werte 'feld(step)' und 'feld(step+1)'. Da das eigentliche Array durch seinen Namen 'flugx' als Parameter übergeben werden soll, muß das tatsächliche Feld mit 'eval' ermittelt werden. Zu den ermittelten Werten wird noch der jeweilige Offset addiert.

```

else
{
    x=offsx+eval(flug+"["+step+"]");
    y=offsy+eval(flug+"["+step+1]+");
}

```

Um testen zu können, ob die Bewegung schon zu Ende ist, wird noch die Länge des Feldes ermittelt und anschließend die neu ermittelte Position wieder browserspezifisch an die Ebene übergeben:

```

l=eval(flug+".length");

if (document.layers)
{
    document.layers[was].left=x; document.layers[was].top=y;
}
else
{
    document.all[was].style.left=x;
    document.all[was].style.top=y;
}

```

Jetzt wird die Variable 'step' zur nächsten Position im Feld gesetzt. Je nach dem Wert von 'dir' bedeutet dies eine Addition oder Substraktion entsprechend einer Vorwärts- oder Rückwärtsbewegung. Dementsprechend wird auch für beide Richtungen getestet, ob das Bewegungsende erreicht ist.

Bei 'repeat' soll die Bewegung wiederholt werden, 'step' als wieder an den Anfang gesetzt. Es hängt aber von der Bewegungsrichtung ab ob 'step' auf Null oder das

Ende des Feldes gesetzt wird. Bei 'loop' wird 'dir' negiert und die Richtung umgekehrt. Ansonsten wird durch 'return' die Bewegung beendet.

Anschließend wird die Funktion durch setTimeout wieder aufgerufen:

```
step+=dir;
if (step>=1-dir || step<=0)
if (mode=="repeat") step=dir>0?0:1;
else if(mode=="loop") dir=-dir;
else return;
}
}

setTimeout
('fliegen("' + was + '", "' + feld + '", ' + step + ', ' + dir + ', ' + ofsx + ', ' + ofsx + ', ' + mode + '", ' + wait + ')", delay);}
```

Jetzt wird nur noch eine Initialisierungsfunktion benötigt, um die verschiedenen Ebenen zu starten:

```
fliegen("Ebene1", "Flug1", 0, 10, 0, 20, "repeat", 50);
fliegen("Ebene2", "Flug2", .....;)}
```

Hardware + Layout

Um dem Benutzer eine akzeptable Darstellung zu bieten, müssen Sie eigentlich wissen, welche Bildschirmgröße und Auflösung er benutzt. Sonst kann es Ihnen passieren, daß absolut positionierte Teile Ihrer Seite einfach außerhalb des Anzeigebereiches liegen und er gar nicht mitbekommt, was Sie ihm bieten wollen.

Wenn Sie wollen, daß auch User ohne 4er-Browser die Inhalte Ihrer Seite lesen können, müssen Sie eine Alternative anbieten: Seiten ohne DHTML, für geringere Auflösung geschrieben, ohne Frames und ohne JavaScript.

Wer surft denn da?

Um den Besucher sinnvoll leiten zu können, ist es notwendig seinen den Namen des Browsers, die Version des Browsers und seinen Codenamen:

```
<SCRIPT LANGUAGE="JavaScript">

<!-- Versteck Dich vor alten Browsern
```

```

function whatBrowser()
{
    /* Name des Browsers */
    document.Browser.Name.value=navigator.appName;

    /* Versionsnummer des Browsers */
    document.Browser.Version.value=navigator.appVersion;

    /* Codename des Browsers */
    document.Browser.Code.value=navigator.appCodeName;

    /* Header des User Agents */
    document.Browser.Agent.value=navigator.userAgent;
}

// -->

</SCRIPT>

```

Das obige Script ergibt z.B. für Microsofts Internet Explorer 4.01 folgende Angaben

Browser Name:	Microsoft Internet Explorer
Browser Version:	4.0 (compatible; MSIE 4.01; Windows 95; QXW03006)
Browser Code Name:	Mozilla
User-Agent:	Mozilla/4.0 (compatible; MSIE 4.01; Windows 95; QXW03006)

, der Netscape Communicator liefert hingegen

Browser Name:	Netscape
Browser Version:	4.5 [en]C-QXW03101 (Win95; I)
Browser Code Name:	Mozilla
User-Agent:	Mozilla/4.5 [en]C-QXW03101 (Win95; I)

Eine einfache Weiterleitung entsprechend des benutzten Browsers könnte daher wie folgt aussehen:

```

<html>
<head>
<SCRIPT LANGUAGE="JavaScript">

```

```

<!-- Begin
function whatBrowser()
{
if((navigator.appName == "Netscape") &&
    (parseInt(navigator.appVersion) >= 4 ))
    /* Netscape Version größer gleich 4.0 */
    window.location.href="ns.html";
else if ((navigator.appName == "Microsoft Internet Explorer")
    && (parseInt(navigator.appVersion) >= 4 ))
    /* Internet Explorer Version größer gleich 4.0 */
    window.location.href="ie.html";
else
    window.location.href="normal.htm";
}
// End -->
</SCRIPT>
<title>Umleitung</title>
</head>

<BODY onLoad="whatBrowser()">
Ihr Browser unterst&uuml;tzt leider keine Frames.<br>
Ich habe <A HREF="normal.htm">Seiten</A> ohne Frames und ohne
DHTML / JavaScript vorbereitet.<br>
Sie k&ouml;nnen allerdings auch einen modernen (f&uuml;r
private Zwecke kostenlosen) Browser von Microsoft bzw.
Netscape den<ul>
<li><A HREF="http://www.microsoft.com">Microsoft Internet
Explorer</A></li>
<li><A HREF="http://www.netscape.com">Netscape
Navigator/Communicator.</A></li>
</ul>
Damit k&ouml;nnen sie meine Seiten vollst&auml;ndig
betrachten.<br>
</BODY>

```

```
</html>
```

Plugins

Es gibt für die verschiedenen Browser die unterschiedlichsten Erweiterungen (= „Plugins“). Diese einzusetzen macht nur Sinn, wenn der Besucher sie installiert hat. Daher gehört zu einer guten WWW-Seite die Überprüfung, welche Plugins installiert sind.

Auch hier hilft JavaScript:

```
<SCRIPT LANGUAGE="JavaScript">
<!--
/* Anzahl installierter Plugins ermitteln */
numPlugins = navigator.plugins.length;
if (numPlugins > 0)
{
    /* Was denn, kein Plugin istanlliert */
    document.writeln("<b><font size=+3>Installed plug-
        ins</font></b><br>");
}
else
{
    document.writeln("<b><font size=+2>No plug-ins are
        installed.</font></b><br>");
}
/* Ausgabe der installierten Plugins in den Browser */
for (i = 0; i < numPlugins; i++)
{
    plugin = navigator.plugins[i];
    /* Ausgabe Name , zentriert, fett, größere Schrift */
    document.write("<center><font size=+1><b>");
    document.write(plugin.name);
    document.writeln("</b></font></center><br>");

    /* Wo befindet sich die Datei */
    document.writeln("<dl>");
```

```

document.writeln("<dd>File name:");
document.write(plugin.filename);
document.write("<dd><br>");

/* kurze Beschreibung, was der Hersteller halt so sagt */
document.write(plugin.description);
document.writeln("</dl>");
document.writeln("<p>");

/* Anlegen einer Tabelle:
    • Mime-Typ (Mime Type),
    • Beschreibung (Description),
    • Dateiendung (Suffixes),
    • steht zur Verfügung (Enabled)
*/

document.writeln("<table width=100% border=2
cellpadding=5>");
document.writeln("<tr>");
document.writeln("<th width=20%><font size=-1>Mime
Type</font></th>");
document.writeln("<th width=50%><font size=-
1>Description</font></th>");
document.writeln("<th width=20%><font size=-
1>Suffixes</font></th>");
document.writeln("<th><font size=-1>Enabled</th>");
document.writeln("</tr>");
numTypes = plugin.length;
for (j = 0; j < numTypes; j++) {
mimetype = plugin[j];
if (mimetype) {
enabled = "No";
enabledPlugin = mimetype.enabledPlugin;
if (enabledPlugin && (enabledPlugin.name == plugin.name))
enabled = "Yes";

```

```

document.writeln("<tr align=center>");
document.writeln("<td>");
document.write(mimetype.type);
document.writeln("</td>");
document.writeln("<td>");
document.write(mimetype.description);
document.writeln("</td>");
document.writeln("<td>");
document.write(mimetype.suffixes);
document.writeln("</td>");
document.writeln("<td>");
document.writeln(enabled);
document.writeln("</td>");
document.writeln("</tr>");
}
}
document.write("</table>");

/* bitte etwas Abstand und einen Strich vor dem nächsten
Plugin */
document.write("<p><hr><p>");
}
// -->
</SCRIPT>

```

Eine Beispiel für die Ausgabe:

RealPlayer(tm) LiveConnect-Enabled Plug-In (32-bit)

File name: E:\INTERNET\NETSCAPE\COMMUNICATOR\PROGRAM\plugins\Npra32.dll

RealPlayer(tm) LiveConnect-Enabled Plug-In

Mime Type	Description	Suffixes	Enabled
audio/x-pn-realaudio-plugin	RealPlayer(tm) as Plug-in	rpm	Yes

Headspace Beatnik Player Stub V1.0.0.1

File name: E:\INTERNET\NETSCAPE\COMMUNICATOR\PROGRAM\plugins\NPBeatSP.dll

Headspace Player Stub for Netscape Communicator

Mime Type	Description	Suffixes	Enabled
audio/x-rmf	RMF	rpm	Yes
audio/rmf	RMF	rpm	Yes

QuickTime Plug-In

File name: E:\INTERNET\NETSCAPE\COMMUNICATOR\PROGRAM\plugins\npqtplugin.dll

QuickTime Plug-In for Win32

Mime Type	Description	Suffixes	Enabled
image/x-sgi	SGI	sgi	Yes
image/x-photoshop	Photoshop	psd	Yes
image/x-bmp	BMP	bmp	Yes
image/x-macpaint	MacPaint	pnt, pntg	Yes
image/x-targa	Targa	tga, targa	Yes
...

Bildschirm

Der Netscape Navigator hat seit der Version 3.0 die Möglichkeit, die Bildschirmgröße auszulesen:

```
<script language="Javascript">
<!--
// ab Netscape Navigator Version 3.0

function Bildschirm()
{
    var Bildbreite = document.images[0].width;

    if(Bildbreite <= 640)
    {
```

```

        // Auflösung 640 x 480
        location.href = "640.html";
    }

    if(Bildbreite>640 && Bildbreite <= 800)
    {
        // Auflösung 800 x 600
        location.href = "800.html";
    }

    if(Bildbreite>800 && Bildbreite <= 1024)
    {
        // Auflösung 1024 x 768
        location.href = "1024.html";
    }
}

//->
</script>

```

Ab der Version 1.2 von JavaScript besteht die Möglichkeit für alle:

```

<script language="Javascript">
<!--
// ab JavaScript 1.2
function Bildschirm()
{
    var Breite = screen.width;
    var Hoehe = screen.height;

    // Abfrage entsprechend obigen Beispiel
}

//->
</script>

```

Troubleshooting

Beim Verwenden dynamischer Seiteninhalte existiert ein Problem, das schon seit 'document.write' bekannt und dokumentiert ist:

Seiten mit dynamischem Inhalt verlieren ihr 'dynamisches Layout', wenn die Größe des Browserfensters verändert wird.

Ein mit DHTML erstelltes Layout kann nur durch das Neuladen der Seite wiederhergestellt werden, wenn das Browserfenster verändert wurde. Man kann jedoch den 'RESIZE-Event' benutzen und ein Script schreiben zum Neuladen der Seite. Der User muß dann zwar warten, aber das Seitenlayout und die damit zusammenhängende Scripttechnik gehen nicht verloren.

Es können mehrere Wege gewählt werden, um dies zu verwirklichen. Zwei möchte ich Ihnen hier vorstellen.

```
<SCRIPT LANGUAGE="JavaScript1.2">
<!-- auskommentieren
/* onresize erst ab Javascript 1.2 */

function neuladen()
{
    window.location.reload()
}
window.onresize = neuladen;

//-->
</SCRIPT>
```

Und:

```
<BODY onResize="window.location.reload()">
```

Der NAVIGATOR 4.04 beinhaltet jedoch einen undokumentierten onload/onresize-Bug. Jedes Laden der Seite wird interpretiert als Veränderung des Browserfensters. Wird das Fenster verändert und die Seite enthält obiges Script, wird die Seite endlos wiedergeladen (loop).

Um den Loop zu verhindern, muß der 'RESIZE-Handler' immer definiert werden nachdem die Seite geladen wurde. Denn dann gibt es beim Neuladen der Seite auf

die ursprüngliche Größe keinen ONRESIZE-Handler im Konflikt mit ONLOAD bis die Seite vollständig stabil erscheint.

```
<SCRIPT LANGUAGE="JavaScript1.2">
<!-- auskommentieren
function neuladen()
{
    window.location.reload()
}

function setResize()
{
    setTimeout("window.onresize=neuladen",500);
}
window.onload = setResize;
//-->
</SCRIPT>
```

Mit dem Script wird die Seite geladen und die 'setResize-Funktion' aufgerufen, die in der Folge 'onresize' setzt. Mit dem Benutzen der setTimeout-Methode wird der Handler eine halbe Sekunde nach dem vollständigen Laden der Seite aufgerufen.

Wenn 'onload' schon benutzt wird zur Scriptinitialisierung oder für eine andere Funktion, muß lediglich folgende Zeile hinzugefügt werden:

```
setTimeout("window.onresize=neuladen",500);
```

Da 'onresize' nur mit dem Navigator 4 funktioniert und nicht mit dem IE, sollte zwecks Crossbrowser-Kompatibilität das vollständige Script folgendermaßen aussehen:

```
<SCRIPT LANGUAGE="JavaScript1.2">
<!-- auskommentieren
NS4 = /document.layers);
function neuladen()
{
    window.location.reload()
}

function setResize()
```

```

{
    setTimeout("window.onresize=neuladen",500);
}
if (NS4)
    window.onload = setResize;
//-->
</SCRIPT>

```

Wie Sie sehen, tauchen beim Arbeiten mit DHTML immer wieder neue Probleme auf, die öffentlich mit der nächsten Browsergeneration gelöst und durch andere ersetzt werden.

Links ins WWW

- Das W3C - die Instanz für HTML im Web
<http://www.w3.org/>
- Milch&Zucker - die DHTML-Seite von Bergmann/Gamperl
<http://dhtml.seite.net/>
- Webcoder.com - Your home for Javascript and Dynamic HTML
http://www.webcoder.com/index_real.html
- Webreference.com/dhtml - Dynamic HTML Lab
<http://www.webreference.com/dhtml/>
- The DHTMLzone from Macromedia
<http://www.dhtmlzone.com/index.html>
- Hotwired Webmonkey - DHTML Tutorial
http://www.hotwired.com/webmonkey/98/10/index0a.html?tw=dynamic_html
- Die HTML Writers Guild
<http://www.hwg.org/>
- Das Netscape Entwickler Center
<http://developer.netscape.com/>