PROGRAMMER'S CHOICE

Buchtipp

Stefan Münz

Professionelle Websites

Programmierung, Design und Administration von Webseiten

2., überarbeitete Auflage

HTML, CSS, JavaScript, DOM, PHP, MySQL
XML, XSLT, SOAP, RSS, Apache, Linux

ADDISON-WESLEY

Mit W3C-Recommendation, Dokumentationen, Software-Tools

CD

6 Erweiterte Features von HTML und CSS

Einige Aspekte von HTML und CSS wurden in Kapitel 4 bewusst nicht behandelt, weil es sich um problembehaftete oder speziellere Features handelt. In diesem Kapitel gehen wir gesondert darauf ein.

6.1 Mehrfenstertechnik (Frames)

Als der damals führende Browser-Anbieter Netscape 1995 die Technik der Frames implementierte und dazu eine Reihe nicht standardisierter HTML-Elemente einführte, wurde das neue Feature von der Gemeinde der Webdesigner begeistert aufgenommen. Endlich war es möglich, beispielsweise links eine fixe Navigation zu haben, während sich rechts der scrollbare Inhalt befand. Die Navigation befand sich in einer separaten Datei und musste nicht in jeder Datei neu notiert werden.

So weit ist der Segen der Mehrfenstertechnik gut nachvollziehbar. Dennoch tat sich das W3-Konsortium schwer damit, die Frames in den HTML-Standard mit aufzunehmen. 1998 übernahm man das Konzept zwar in HTML 4.0, führte dazu aber eine eigene HTML-Variante neben »strict« und »transitional« ein: die Variante »Frameset«.

Mittlerweile gelten Frames in professionellen Kreisen als verpönt. Dafür gibt es mehrere Gründe:

- Inhaltsseiten haben selbst keine vollständige Struktur und in der Regel keine Navigationslinks.
- Anwender können z. B. keine Bookmarks auf bestimmte Unterseiten eines Angebots setzen, weil in der Adresszeile des Browsers immer nur der URI der Seite angezeigt wird, die das Frameset enthält. Durch etwas Programmieraufwand ist dieses Problem zwar lösbar, doch die meisten Anbieter, die Frames einsetzen, kümmern sich nicht darum.
- Frames sind ungünstig für Suchmaschinen. Die meisten Suchmaschinen lösen zwar die Frame-Struktur auf und indexieren die inhaltstragenden Seiten. Doch die Links bei Suchtreffern führen dann direkt zu den Unterseiten. Da diese aber nur einen Teil der Seite darstellen und meist keine eigene Navigation enthalten, landen die Seitenbesucher in einer Hypertext-Sackgasse, die oft auch vom Layout her wie eine »halbe Sache« wirkt. Der Besucher erhält keinen besonders guten Eindruck vom Anbieter der Seite.

- Sind Frames bei der Bildschirmanzeige noch praktisch, so stellen sie beim Ausdrucken von Webseiten nicht selten ein Problem dar. Hat der Anwender den Fokus im Frame für die Navigation, kann es beispielsweise passieren, dass der Browser nur die Navigation ausdruckt und nicht wie gewünscht den Inhalt. Unerfahrene Anwender sind dadurch schnell irritiert.
- Bei drei und mehr Frames soll pro Link nicht selten mehr als ein anderer Frame seinen Inhalt ändern. Mit HTML allein ist es jedoch nicht möglich, zwei oder mehr URIs pro Link zu adressieren. Deshalb wird zu JavaScript gegriffen, was aber nicht funktioniert, wenn der Anwender JavaScript ausgeschaltet hat. Die gesamte Navigation funktioniert dann nicht mehr.
- Frame-basierte Seiten sind ein Problem für Browser oder Ausgabegeräte, die nicht frame-fähig sind. Zwar erkennen modernere nicht grafische Browser wie Lynx Frames, doch das Handling für den Anwender ist meist unbefriedigend.
- Frames erfordern mehr Kommunikation zwischen Browser und Webserver, da pro Frame Ressourcen übertragen werden müssen.
- Frames verlocken zu einem unfairen, urheberrechtlich sehr bedenklichen Trick: Nur die Navigation gehört zur eigenen Site, während die Inhalte aus fremden Quellen stammen. In der Adresszeile des Browsers ist das für den Anwender jedoch nicht erkennbar.

Im Grunde gibt es zwei Hauptgründe, weswegen Frames immer noch zum Einsatz kommen:

- Bestimmte Inhalte scrollen nicht mit, z. B. eine Navigation, ein Logo usw.
- Wiederkehrende Teile wie der Code für eine Navigation muss nur einmal in einer separaten Datei notiert werden, aber nicht mehr in jeder einzelnen Datei.

Für »non-scrolling-regions« steht mit der CSS-Angabe `position:fixed` mittlerweile eine Alternative zur Verfügung. Wir werden in diesem Kapitel auch zeigen, wie Sie diese Alternative richtig einsetzen.

Gegen das Argument der einmaligen Notation von Code wird von Frames-Gegnern in der Regel angeführt, dass es serverseitige Möglichkeiten gebe, Code nicht immer wieder kopieren zu müssen. Dies läuft jedoch auf dynamisch erzeugte Seiten hinaus. Im einfachsten Fall über Server Side Includes, meist aber über Scriptsprachen wie PHP. Wenn Sie ohnehin eine Site entwickeln, die nicht mit statischen HTML-Dateien, sondern etwa mit PHP-Scripts arbeitet, dann sind Frames ohnehin nur lästig und problematisch. Wenn Sie ein HTML-basiertes Projekt dagegen unter anderem auch auf Wegen wie CD-ROM verbreiten möchten, müssen Sie statische HTML-Dateien verwenden. In diesem Fall aber zieht das Argument der Frames-Gegner nicht und es ist unter Umständen durchaus vertretbar, sich bewusst für den Einsatz von Frames zu entscheiden.

Abbildung 6.1: Frameset im grafischen Browser (oben) und in Lynx (unten)

Link zum Thema Frames-Problematik:

Artikel »Warum Frames out sind«:
http://www.subotnik.net/html/frames.html

6.1.1 HTML mit Framesets und Frames

Das »Denken in Frames« erfordert zunächst einmal, sich Gedanken über mehrere unterschiedliche Dateien zu machen. Erforderlich sind eine Datei, in der das Frameset definiert wird, also die Aufteilung der Einzelfenster sowie je eine Datei bzw. Quelle, die beim Aufruf der Frameset-Datei in die einzelnen Frame-Fenster geladen werden soll.

Datei mit Frameset-Definitionen

Das nachfolgende Listing zeigt den kompletten Quelltext einer Datei mit einer Frameset-Definition:

Listing 6.1: Datei mit Frameset-Definition

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
   "http://www.w3.org/TR/html4/frameset.dtd">
<html>
   <head>
      <title>Mein erstes Frameset</title>
   </head>
   <frameset cols="250,*">
      <frame src="verweise.htm" name="Navigation">
      <frame src="startseite.htm" name="Daten">
      <noframes>
         Ihr Browser kann diese Seite leider nicht anzeigen!
      </noframes>
   </frameset>
</html>
```

Zunächst fällt auf, dass eine andere Dokumenttyp-Deklaration verwendet wird. In Dateien, die ein Frameset definieren, müssen Sie diese Deklaration verwenden. Damit erfüllt eine solche Seite den HTML 4.01-Standard in der Variante für Framesets.

Die weitere Struktur weicht ebenfalls stark von sonst üblichen HTML-Dokumenten ab. Das html-Element enthält nicht wie sonst üblich die Elemente `head` und `body` als Inhalt, sondern die Elemente `head` und `frameset`. Das body-Element wird also durch das frameset-Element ersetzt. Im head-Element können jedoch die gleichen Daten stehen wie bei gewöhnlichen HTML-Dokumenten. Das title-Element ist Pflicht und andere Angaben, wie etwa meta- oder link-Elemente, sind ohne Ausnahme erlaubt und meist auch sehr sinnvoll.

Das frameset-Element (`<frameset>...</frameset>`) bestimmt die Verteilung der Fenster. Dazu kann es die beiden Attribute `rows=` und `cols=` enthalten. Durch das rows-Attribut werden Fenster untereinander angeordnet (***rows*** = Reihen), durch das cols-Attribut nebeneinander (cols = ***columns*** = Spalten).

Im obigen Beispiel-Listing werden durch die Angabe `cols="250,*"` zwei Frame-Fenster nebeneinander angeordnet. Das linke Frame-Fenster soll 250 Pixel breit sein. Das rechte soll so breit sein, wie der Rest des Browser-Anzeigefensters es erlaubt. Die Wertzuweisung besteht also in kommagetrennten Werten, wobei eine Zahl eine Pixelangabe bedeutet. Ferner sind Prozentangaben erlaubt wie etwa `cols="30%,*"`. Das Sternchen ist ein Platzhalter und bedeutet »die Breite soll vom Browser bestimmt werden«.

Selbstverständlich ist eine Aufteilung in mehr als zwei Spalten oder Reihen möglich. So definiert beispielsweise `rows="200,250,*"` drei Frame-Fenster untereinander, wobei das erste 200 Pixel hoch ist, das zweite 250 Pixel hoch und das dritte soll so hoch sein wie der Rest der Anzeigehöhe des Browser-Fensters es ermöglicht.

Möglich sind auch Angaben wie `cols="100,3*,5*"`. Dies ist sinnvoll, wenn Sie ein oder zwei Frames mit einer Pixelangabe zu Höhe und Breite versehen möchten, die übrigen Höhen bzw. Breiten dagegen flexibel gestalten. Die Angabe im Beispiel bedeutet: drei Frame-Fenster nebeneinander, wobei das erste 100 Pixel breit sein soll. Von der verbleibenden Breite soll

das zweite Fenster drei Achtel einnehmen und das dritte fünf Achtel. Zahlen, gefolgt von einem Sternchen, werden als Verhältnisangabe betrachtet, ähnlich wie Prozentangaben, jedoch bezogen auf den unbekannt großen, verbleibenden Teil des Browser-Fensters. Die Summe der Zahlen ist der gemeinsame Nenner. Die entstehenden Bruchwerte sind für den Browser die Vorgabewerte zur Berechnung der tatsächlichen Framefenster-Breite bzw. -Höhe.

Die Attribute `cols=` und `rows=` dürfen auch beide gleichzeitig verwendet werden. Ein Beispiel:

```
<frameset  rows="100,*" cols="25%,*">
   <frame src="obenlinks.html" name="f1">
   <frame src="obenrechts.html" name="f2">
   <frame src="untenlinks.html" name="f3">
   <frame src="untenrechts.html" name="f4">
</frameset>
```

Durch eine solche Aufteilung wird ein Gitter mit so vielen Spalten mal Reihen wie angegeben erzeugt. Häufig werden jedoch auch andere Aufteilungen gewünscht, wie etwa »oben ein Streifen über die gesamte Breite und unten zwei Spalten«. Für solche Aufteilungen besteht die Möglichkeit, Frameset-Definitionen zu verschachteln:

```
<frameset rows="100,*">
   <frame src="oben.html" name="f1">
   <frameset rows="250,*">
      <frame src="unenlinks.html" name="f2">
      <frame src="untenrechts.html" name="f3">
  </frameset>
</frameset>
```

Es werden also ein äußeres und ein inneres Frameset definiert. Das äußere Frameset besorgt mit `rows=` die Aufteilung in zwei Frame-Fenster untereinander. Das frame-Element für die obere Quelle wird auch erst einmal notiert. Anstelle des frame-Elements für das untere Frame-Fenster wird jedoch ein weiteres, inneres frameset-Element notiert. Dieses bewirkt eine Aufteilung in zwei weitere, nebeneinander anzuordnende Frame-Fenster (`cols=`).

Für jeden kommaseparierten Wert, der bei einem rows- oder cols-Attribut zugewiesen wird, muss innerhalb des frameset-Elements also entweder ein frame-Element notiert werden, welches den Fensterinhalt angibt, oder ein weiteres, inneres frameset-Element, welches den Raum des Frame-Fensters für eine Aufsplittung in weitere Frame-Fenster nutzt.

Während das frameset-Element aus Anfangs- und End-Tag besteht, ist `<frame...>` ein Standalone-Tag. In XHTML muss es folglich in der Form `<frameset ... />` notiert werden. Das frame-Element hat zwei wichtige Standardattribute. Beim src-Attribut wird der URI des Inhalts angegeben, der im entsprechenden Frame-Fenster angezeigt werden soll. Es kann sich um eine lokal referenzierte Datei handeln oder auch um eine entfernte, absolute Internetadresse. Inhalte können HTML-Dateien sein, aber natürlich auch andere Quellen, wie Grafiken, Flash-Movies, PDF-Dateien oder Ähnliches. Auch dynamische Quellen wie

PHP-Scripts können selbstverständlich in jedem einzelnen Frame-Fenster als Quelle angegeben werden. Bei Grafik- oder Multimedia-Quellen ist allerdings zu beachten, dass hierbei keine Möglichkeit besteht, einen Alternativtext anzugeben.

Das andere wichtige Attribut für jedes frame-Element ist das name-Attribut. Der frei vergebbare Name ist vor allem wichtig, wenn Seiten, die innerhalb eines Frame-Fensters angezeigt werden, mit einem Link den Inhalt eines anderen Frame-Fensters verändern wollen. Bei solchen Links muss das andere Frame-Fenster unter dem hier vergebenen Namen mit adressiert werden.

Noframes-Bereiche

Zwischen `<frameset>` und `</frameset>` darf neben inneren frameset-Elementen und frame-Elementen noch ein weiteres Element stehen, ausgezeichnet durch `<noframes>...</noframes>`. Dieses Element erlaubt es, für Browser oder andere Client-Programme, welche keine Frames anzeigen können, einen alternativen Inhalt anzubieten. Ein Beispiel:

```
<frameset rows="100,*">
   <frame src="oben.html" name="f1">
   <frameset rows="250,*">
      <frame src="unenlinks.html" name="f2">
      <frame src="untenrechts.html" name="f3">
  </frameset>
  <noframes>
    <h1>Liebe Besucher</h1>
    <p>
       Diese Site verwendet Frames.
       Bitte rufen Sie die <a href="noframes.html">
       Navigation für nicht-frame-fähige
       Browser</a> auf!
    .</p>
  </noframes>
</frameset>
```

Das noframes-Element darf am Ende einer frameset-Struktur vor dem schließenden `</frameset>`-Tag des äußersten Framesets notiert werden. Als Inhalt sind beliebige Block- und Inline-Elemente erlaubt.

Interessant ist, dass das noframes-Element auch in HTML-Dokumenten notiert werden darf, die innerhalb eines Framesets in einem Frame-Fenster angezeigt werden. Dort muss allerdings im Dokumenttyp die Transitional-Variante von HTML angegeben werden. Das noframes-Element kann dann als ganz normales Element im Dateikörper notiert werden. Innerhalb davon können z.B. Rückverweise auf eine Seite mit Navigationslinks notiert werden. Browser wie Lynx, die zwar Framesets erkennen, jedoch nicht anzeigen können, stellen den Inhalt von noframes-Elementen im Dateikörper dar. Browser, die Frames anzeigen können, ignorieren die Inhalte dagegen.

6.1.2 Links zu anderen Frame-Fenstern

Wie bereits erwähnt, müssen Links innerhalb eines Framesets, deren Ziel in einem anderen als dem aktuellen Frame-Fenster angezeigt werden soll, das gewünschte Frame-Fenster benennen. Anzugeben ist der Name, der bei `<frame ... name=>` vergeben wurde. Ein Beispiel:

```
<a href="index.htm" target="inhalt">Startseite</a>
```

Dieser Link öffnet sein Ziel in einem Fenster, das mit `<frame ... name="inhalt">` definiert wurde. Das target-Attribut gibt den gewünschten Fensternamen an (*target* = Ziel).

Bei Navigationslinks, die alle auf ein anderes Frame-Fenster verweisen, gibt es jedoch noch eine andere Möglichkeit, nämlich im Kopfbereich des HTML-Dokuments ein Zielfenster vorzugeben. Beispiel:

```
<head>
   <!-- andere Kopfdatendefinitionen -->
   <base target="inhalt">
</head>
```

Das base-Element kann ebenfalls ein target-Attribut enthalten. So notiert wie im Beispiel hat es die Wirkung, dass alle Linkziele des aktuellen Dokuments in einem Frame-Fenster namens `inhalt` geöffnet werden.

Groß- und Kleinschreibung werden übrigens unterschieden. Wenn beispielsweise ein Fenster mit `<frame ... name="Inhalt">` definiert wurde, kann es nicht mit `<a ... target="inhalt">` angesprochen werden! target-Angaben zu nicht existierenden Frame-Fenstern bewirken in den meisten Browsern übrigens, dass der Link stattdessen in einem neuen Fenster oder in einem neuen Tab geöffnet wird.

Wichtig zu wissen ist ferner, dass HTML-Dokumente, die in Hyperlinks oder im base-Element das target-Attribut benutzen, in der Dokumenttyp-Deklaration nicht mit der HTML-Variante »strict« ausgezeichnet werden dürfen. Das target-Attribut gehört zu den HTML-Bestandteilen, die im »eigentlich gewünschten« HTML keinen Platz mehr haben, ebenso wenig wie Frames allgemein. Verwenden Sie für Dokumente, in denen target-Attribute vorkommen, stattdessen die HTML-Variante »transitional«, also so:

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

6.1.3 Rahmen und andere Frame-Eigenschaften

Normalerweise zeigen frame-fähige Browser zwischen den Frame-Fenstern systemabhängig aussehende Rahmen an, die es dem Anwender unter anderem erlauben, das Fensterverhältnis durch Ziehen der Rahmen mit der Maus zu ändern. Ein häufiger Wunsch von Seitenanbietern sind »unsichtbare Fensterrahmen«. Dadurch erscheint dem Anwender die gesamte Seite wie aus einem Dokument, obwohl in Wirklichkeit ein Frameset am Werk ist.

Meist wurde dieses Feature in der Vergangenheit deshalb verlangt, um zwar eine nahtlose Ansicht im Browser-Fenster anzubieten, jedoch mit einem fixen Bereich für Logo, Navigation usw., der nicht mit scrollt. Der Wunsch ist nachvollziehbar, doch das Mittel, hierfür zu einem Frameset zu greifen, ist mittlerweile nicht mehr zeitgemäß. Fixe Bereiche sollten mithilfe von CSS definiert werden, wie in ***Abschnitt 6.1.4*** beschrieben.

Generell wird die Rahmendicke wie im folgenden Beispiel bestimmt:

```
<frame src="index.htm" name="inhalt" frameborder="10">
```

Das frameborder-Attribut ist laut HTML-Spezifikation in `<frame>`-Tags erlaubt. Dort notiert, bestimmt es die Rahmendicke in Pixel, und zwar für alle Rahmen zwischen diesem Frame-Fenster und benachbarten Frame-Fenstern. Um keine Rahmen anzuzeigen, wird der zugewiesene Wert auf 0 gesetzt. Notieren Sie die frameborder-Angabe sicherheitshalber in allen `<frame>`-Tags. Andernfalls erreichen Sie in den Browsern möglicherweise nicht den gewünschten Effekt.

Ein anderer Fall ist, dass Sie die Rahmen zwischen den Frame-Fenstern zwar optisch für sinnvoll halten, aber nicht möchten, dass der Anwender die Größe der Frame-Fenster damit verändern kann. Für diesen Fall gibt es ein weiteres Attribut, wie das nachfolgende Beispiel zeigt:

```
<frame src="index.htm" name="inhalt" noresize>
```

Das Standalone-Attribut `noresize` (also »kein Resize«, keine Größenänderung) verhindert für die Rahmen um das betroffene Frame-Fenster, dass diese mit der Maus in ihrer Position veränderbar sind. Beachten Sie, dass Sie bei XHTML `noresize="noresize"` notieren müssen.

Schließlich können Sie noch das Scrollverhalten von Frame-Fenstern beeinflussen. Dazu können Sie im `<frame>`-Tag eines Fensters das Attribut `scrolling=` wahlweise mit den Werten `yes`, `no` oder `auto` versorgen: `yes` erzwingt Scrollleisten sogar dann, wenn der Inhalt gar kein Scrollen erfordert; `no` verhindert Scrollleisten in jedem Fall, auch dann, wenn der Inhalt Scrollen erfordert, und `auto`, die Voreinstellung, bewirkt, dass Scrollbars dann angezeigt werden, wenn es der Inhalt erfordert.

Auf weitere Eigenschaften gehen wir an dieser Stelle nicht mehr ein, da diese entweder nicht zum HTML-Standard gehören oder nicht mehr zeitgemäß sind.

6.1.4 Fixe Bereiche ohne Frames

Wie bereits erwähnt, werden Frames in den meisten Fällen nur eingesetzt, um einen fixen Bereich zu haben, der nicht mit scrollt (auch als ***Non-Scrolling-Region*** bezeichnet). In diesem Abschnitt möchten wir zeigen, wie so etwas auch ohne Frames funktioniert – aus der Überlegung heraus, dass es einfach zu schade ist, die zahlreichen Nachteile von Frames in

Kauf zu nehmen, wenn das gewünschte Ziel auch anders erreichbar ist. Das HTML-Dokument hat folgenden Inhalt:

Listing 6.2: HTML-Dokument mit Bereichen für Navigationslinks und Inhalt

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
   <head>
      <meta http-equiv="content-type" content="text/html;
            charset=ISO-8859-1">
      <title>Fixer Bereich</title>
      <link rel="stylesheet" type="text/css" href="fix.css">
   </head>
   <body>
      <div id="navigation">
         <a href="s2.html">Seite 2</a><br>
         <a href="s3.html">Seite 3</a><br>
         <a href="s4.html">Seite 4</a>
      </div>
      <div id="content">
         <h1>Seite 1</h1>
         <p>viel Inhalt</p> ... <p>viel Inhalt</p>
      </div>
   </body>
</html>
```

Die beiden Bereiche erhalten die id-Namen `navigation` und `content`. In der eingebundenen CSS-Datei ***fix.css*** werden folgende Formate definiert:

Listing 6.3: CSS-Formatdefinition für fixen und normalen Bereich

```
#navigation {
   position:fixed;
   top:20px;
   left:20px;
   padding-right:20px;
   border-right:blue solid 4px;
}
#content {
   margin-left:200px;
   margin-right:20px;
}
```

Der Bereich `navigation` wird mit der Angabe `position:fixed` fest positioniert. Beim Scrollen wandert sein Inhalt nicht mit. Die restlichen Definitionen betreffen Randabstände sowie einen blauen dicken Rahmen rechts von den Links.

Der Bereich `content` soll im normalen Textfluss bleiben, erhält also keine Angabe zur Positionierung. Mit `margin-left:200px` wird ein hinreichend großer linker Rand geschaffen, der die Anzeige der Navigationslinks in diesem Bereich erlaubt.

Eigentlich ist das bereits alles und in Browsern, welche `position:fixed` interpretieren, funktioniert auch alles wie gewünscht.

Abbildung 6.2: Fixer Bereich im Firefox-Browser

Der Inhalt kann gescrollt werden, während die Links fest an ihrer Position bleiben. Leider interpretiert der MS Internet Explorer bis einschließlich Version 6.0 `position:fixed` nicht. Stattdessen ignoriert er die Positionierungsangabe, was durchaus ein korrektes Verhalten ist, aber dazu führt, dass die beiden Bereiche beide im normalen Textfluss verbleiben.

Abbildung 6.3: Nicht erkannter fixer Bereich im MS Internet Explorer

Der div-Bereich für die Navigationslinks verhält sich also wie ein gewöhnliches Blockelement im Textfluss. Es steht an der Stelle, wo es notiert wird (zuerst), und nimmt die gesamte verfügbare Breite ein, so dass der div-Bereich für den Inhalt erst unterhalb davon angezeigt wird.

Dieses Verhalten ist jedoch nicht erwünscht. Andererseits kann ein so verbreiteter Browser nicht einfach ignoriert werden. Wir benötigen also eine Behelfslösung. Dank anderer, proprietärer Fähigkeiten des Internet Explorers ist das Verhalten von `position:fixed` »simulierbar«.

Workaround für den Internet Explorer

Dazu wird zunächst der Inhalt der CSS-Datei *fix.css* wie folgt geändert:

```
body > #navigation {
   position:fixed;
   top:20px;
   left:20px;
   padding-right:20px;
   border-right:blue solid 4px;
}

body > #content {
   margin-left:200px;
   margin-right:20px;
}
```

Geändert wird die Syntax der Selektoren dahingehend, dass von den CSS-2.0-spezifischen Adressierungsmöglichkeiten für Elemente Gebrauch gemacht wird. Diese werden vom Internet Explorer ebenfalls nicht interpretiert. Als Ergebnis ignoriert dieser Browser die Formatdefinitionen für beide div-Bereiche komplett.

Im zweiten Schritt bekommt der Internet Explorer stattdessen eine nur für ihn lesbare CSS-Datei spendiert. Dazu wird im HTML-Dokument – am besten hinter dem `<link>`-Tag, welches das normale Stylesheet einbindet – Folgendes notiert:

```
<!--[if gte IE 5]>
   <link rel="stylesheet" type="text/css" href="ie-fix.css">
<![endif]-->
```

Der gesamte Code stellt aus HTML-Sicht einen Kommentar dar. Eigentlich sollten Browser den Inhalt von Kommentaren ignorieren und die meisten tun das auch. Doch nicht so der Internet Explorer. Hier kann der Kommentar interpretierbaren Inhalt haben. Das Konstrukt `[if gte IE 5]>.... <![endif]` wird, wenn ohne Leerraum nach bzw. vor den Kommentarklammern notiert, als Verarbeitungsanweisung interpretiert. `[if gte IE 5]>` bedeutet so viel wie: »Wenn ein Internet Explorer mit Versionsnummer größer oder gleich 5.0 das hier liest, dann soll er den HTML-Code bis zu `<![endif]` trotz Kommentar interpretieren.« Genutzt wird diese Fähigkeit in unserem Fall, um mittels `<link>`-Tag eine weitere CSS-Datei speziell für den Internet Explorer einzubinden. Im Beispiel nennen wir sie *ie-fix.css*.

Diese spezielle CSS-Datei erhält folgenden Inhalt:

```
html, body {
   overflow:hidden;
   width:100%;
   height:100%;
}
#navigation {
   position:absolute;
   top:20px;
   left:20px;
   padding-right:20px;
   border-right:blue solid 4px;
}
#content {
   position:absolute;
   top:0px;
   left:200px;
   padding-top:20px;
   height:expression(document.body.clientHeight - 20 + "px");
   width:expression(document.body.clientWidth - 200 + "px");
   overflow:auto;
}
```

Zunächst wird für die Basiselemente `html` und `body` festgelegt, dass diese die volle Breite und Höhe einnehmen. Gleichzeitig wird bestimmt, dass Inhalte, die länger oder breiter sind, einfach abgeschnitten werden (`overflow:hidden`).

Die beiden div-Bereiche mit den id-Namen `navigation` und `content` werden daraufhin beide absolut positioniert. Bis auf die Art der Positionierung sind die Formatdefinitionen für den Bereich `navigation` die gleichen wie in der normalen CSS-Datei.

Bei den Definitionen für `content` gibt es hingegen einige Unterschiede. Mit `top` und `left` muss dessen gewünschte Anfangsposition bestimmt werden, da er ja absolut positioniert wird. Eigentlich soll er 20 Pixel von oben beginnen. Damit der Internet Explorer die vertikale Scrollleiste jedoch wie üblich oben im Fenster beginnen lässt, weisen wir `top:0px` zu und schaffen den gewünschten Abstand nach oben über den Umweg `padding-top:20px`.

Besonders spannend sind die Zuweisungen an die CSS-Eigenschaften `height` und `width`. Anstelle eines festen Werts wird ein Wert zugewiesen, der mithilfe von JScript errechnet wird. Der Internet Explorer erlaubt das Zuweisen gewisser Funktionen an CSS-Eigenschaften, unter anderem die Funktion `expression()`. Diese ermöglicht das Ausführen von JavaScript/JScript-Code. In den beiden Beispielangaben werden die Breite und die Höhe des zu positionierenden Bereichs aus der tatsächlichen Breite und Höhe bestimmt, die der in HTML in diesem Bereich notierte Inhalt einnimmt. Die beiden Objekteigenschaften `document.body.clientWidth` und `document.body.clientHeight` liefern die Werte für den gesamten body-Bereich. Abgezogen werden die gewünschten Startpositionen des Inhaltsbereichs (der Obenwert bei der Höhe und der Linkswert bei der Breite). Ausdrücklich zugewiesen wird dann noch `overflow:auto`. Somit wird der Bereich gescrollt, falls sein Inhalt es erfordert.

Abbildung 6.4: Simulierter funktionsfähiger fixed-Bereich im Internet Explorer

In unserem Beispiel gibt es nun keinerlei Unterschiede in der Funktionalität etwa zwischen Firefox, Opera und anderen einerseits und Internet Explorer andererseits. In allen Browsern bleiben die Navigationslinks fix an ihrer Position.

Wenn Sie den hier beschriebenen Workaround für fix positionierte Bereiche an anderer Stelle, beispielsweise oben, nutzen wollen, müssen Sie gegebenenfalls in der CSS-Datei für den Internet Explorer bei `html` und `body` nicht `overflow:hidden` angeben, sondern `overflow-y:hidden`.

6.1.5 Eingebettete Frames

Nachdem wir im vorherigen Abschnitt eine Möglichkeit beleuchtet haben, um Frames zu vermeiden, werden wir nun noch eine spezielle Form von Frames kennen lernen, die kein Frameset benötigt, sondern direkt im normalen body-Inhalt notiert werden kann.

Eingebettete Frames (auch als *Inline-Frames* bezeichnet) ermöglichen es, an einer gewünschten Stelle innerhalb des sichtbaren Inhalts eines HTML-Dokuments Inhalte aus anderen Dateien oder Quellen einzubinden. Dazu stellt HTML das iframe-Element zur Verfügung (iframe = inline frame). Wie der Name schon vermuten lässt, hat dieses Element Inline-Charakter, d.h., es kann – ähnlich wie Grafikreferenzen – auch mitten im Text notiert werden.

Zum reinen Einbetten einer anderen Quelle kann übrigens auch das object-Element verwendet werden, das im Gegensatz zum iframe-Element zum strict-Standard von HTML gehört (siehe auch ***Abschnitt 4.9.8***). Bei Verwendung des iframe-Elements müssen Sie dagegen den Dokumenttyp für die HTML-Variante »transitional« angeben. Erforderlich ist die Verwendung des iframe-Elements jedoch dann, wenn Hyperlinks ihr Verweisziel im Fenster des Inline-Frames öffnen sollen. Genau diesen Anwendungsfall werden wir im nachfolgenden Beispiel behandeln.

Der Quelltext der Datei mit dem eingebetteten Frame lautet:

Listing 6.4: HTML-Dokument mit eingebettetem Frame-Fenster

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
        "http://www.w3.org/TR/html4/loose.dtd">
<html lang="de">
   <head>
      <meta http-equiv="content-type" content="text/html;
            charset=ISO-8859-1">
      <title>Eingebetteter Frame</title>
   </head>
   <body style="text-align:center">
      <h1>Blindtext</h1>
      <p>
         <a href="seite1.htm" target="content">Seite 1</a> |
         <a href="seite2.htm" target="content">Seite 2</a> |
         <a href="seite3.htm" target="content">Seite 3</a> |
         <a href="seite4.htm" target="content">Seite 4</a>
      </p>
      <p>
         <iframe name="content" src="seite1.htm" frameborder="0"
            style="width:60%; height:200px;
            border:thin solid #COCOCO">
               Ihr Browser unterstützt
               leider keine eingebetteten Frames!
         </iframe>
      </p>
   </body>
</html>
```

Das iframe-Element wird mit Start- und End-Tag notiert. Zwischen `<iframe>` und `</iframe>` kann Inhalt für Browser notiert werden, die das iframe-Element nicht kennen oder interpretieren. Dabei dürfen auch beliebige Block- und Inline-Elemente notiert werden.

Wie beim frame-Element wird auch beim iframe-Element eine Default-Quelle über das src-Attribut referenziert. Es kann sich um einen beliebigen URI handeln, also um eine lokal referenzierte Datei oder auch um eine absolute Internetadresse. Wie auch beim frame-Element ist das name-Attribut wichtig, wenn Linkziele im eingebetteten Frame geöffnet werden sollen.

Weitere Angaben zum Frame-Fenster wie Breite und Höhe des Frame-Fensters werden im obigen Beispiel zeitgemäß über CSS gelöst. Die Angabe `width:60%` im Beispiel bedeutet: 60% der Breite des Elternelements. Dieses ist das umgebende p-Element, welches sich im normalen Textfluss innerhalb des body-Elements befindet. Da es ein Block-Element ist, nimmt es die maximal verfügbare Breite ein, also die gesamte Breite des Anzeigefensters abzüglich der Default-Ränder, die der Browser setzt.

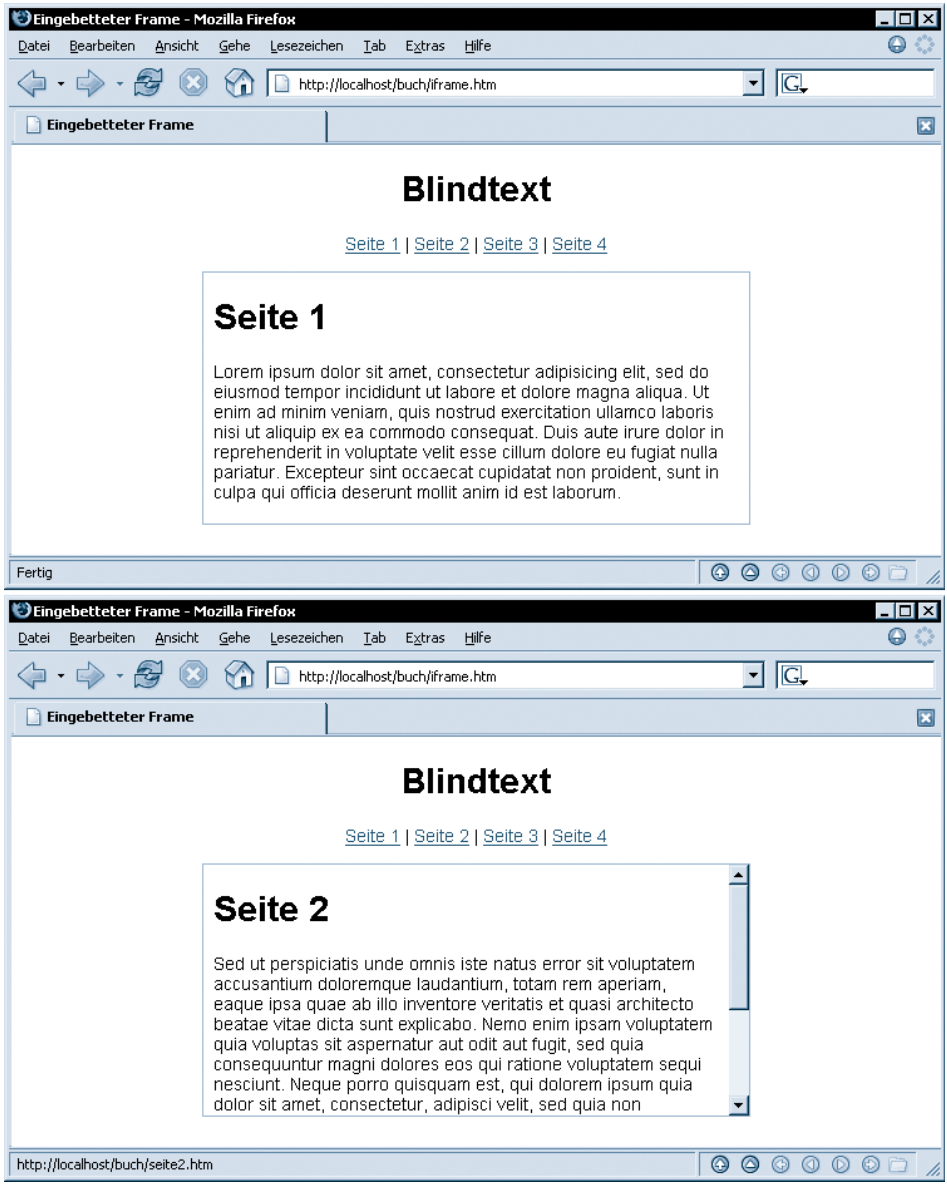

Abbildung 6.5: Im eingebetteten Frame-Fenster angezeigte Verweisziele

Mithilfe von CSS lässt sich auch der Rahmen des eingebetteten Frames optimal gestalten. Im Beispiel haben wir einen dünnen grauen Rahmen (`border:thin solid #COCOCO`) gewählt. Dennoch ist zusätzlich das HTML-Attribut `frameborder="0"` notiert. Damit wird explizit der Frame-eigene Rahmen unterdrückt, den der Internet Explorer andernfalls unabhängig von der CSS-Rahmendefinition anzeigt.

Die Links im Beispiel haben im einleitenden <a>-Tag ein Attribut `target="content"`. Dadurch wird der Fensterbezug zum iframe-Element hergestellt, das mit `name="content"`

definiert wird. Die Verweisziele werden im Frame-Fenster geöffnet. Scrollleisten zeigt der Browser nach Bedarf an. Durch `scrolling="no"` können Sie die Anzeige von Scrollleisten verhindern.

6.2 Automatische Überschriftennummerierung

Insider betonen immer wieder, dass HTML und CSS keine Programmiersprachen seien, sondern Auszeichnungs- bzw. Beschreibungssprachen. Das ist korrekt. Leider hat das Fehlen von operativen Möglichkeiten aber auch Nachteile. Einer davon ist das Problem, einen Nummerierungsalgorithmus etwa für Überschriften anzugeben. Das Problem wurde vom W3-Konsortium registriert und so steht seit CSS2.0 eine Funktionalität zur Verfügung, die es erlaubt, eine individuelle Überschriftennummerierung zu »programmieren«. Doch leider hapert es mit der Browser-Unterstützung. Weder der MS Internet Explorer 6.0 noch der sonst so standardstarke Firefox-Browser 1.0 unterstützen das entsprechende Feature. Lediglich Opera ist dazu in der Lage.

Aus diesem Grund werden wir zwei Vorgangsweisen beschreiben, die sich ergänzen. Zunächst werden wir die in CSS vorgesehene Methode behandeln. Für andere Browser stellen wir anschließend eine Lösung vor, die ein wenig JavasScript-Programmierung erfordert.

6.2.1 Überschriftennummerierung mit CSS

Ein Beispieldokument enthält diverse Überschriften 1., 2. und 3. Ordnung. Diese sollen nummeriert werden. Der HTML-Quelltext lautet:

Listing 6.5: HTML-Dokument mit Überschriften

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
   <head>
      <meta http-equiv="content-type" content="text/html;
            charset=ISO-8859-1">
      <title>Nummerierte Überschriften</title>
      <link rel="stylesheet" type="text/css" href="h-styles.css">
      </style>
   </head>
   <body>
      <h1>Die Fauna (Tierwelt)</h1>
      <h2>Die Avifauna (Vögel)</h2>
      <h3>Laufvögel</h3>
      <h3>Hühnervögel</h3>
      <h3>Gänsevögel</h3>
      <h3>Kranichvögel</h3>
      <h2>Die Entomofauna (Insekten)</h2>
      <h3>Felsenspringer</h3>
      <h3>Fischchen</h3>
```

```
        <h3>Fluginsekten</h3>
        <h2>Die Ichthyofauna (Fische)</h2>
    </body>
</html>
```

Die Überschriften sollen in der Form 1, 1.1, 1.1.1 usw. nummeriert werden. Dazu wird im Beispiel im HTML-Dateikopf eine CSS-Datei namens *h-styles.css* eingebunden. Diese Datei hat folgenden Inhalt:

Listing 6.6: CSS-Datei mit Formatdefinitionen für Überschriftennummerierung

```
h1:before {
    content:counter(level_1) ". ";
    counter-increment:level_1;
    counter-reset:level_2;
}
h2:before {
    content:counter(level_1) "." counter(level_2) " ";
    counter-increment:level_2;
    counter-reset:level_3;
}
h3:before {
    content:counter(level_1) "." counter(level_2) "." counter(level_3) " ";
    counter-increment:level_3;
}
```

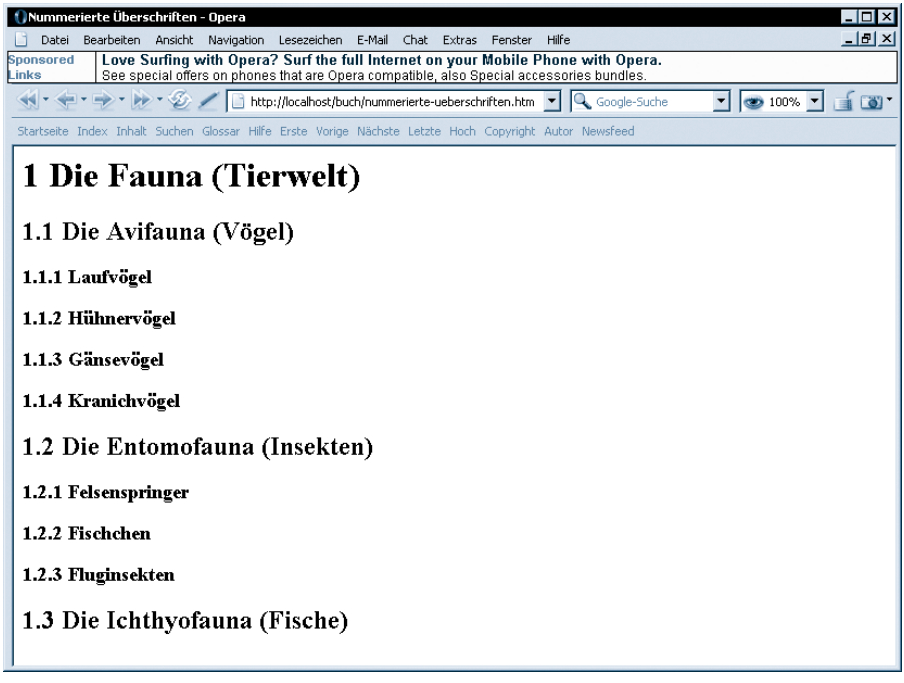

Abbildung 6.6: Mit CSS nummerierte Überschriften im Opera-Browser

Die Überschriftennummerierung benutzt das Pseudoelement `:before`, welches es erlaubt, einen festlegbaren Inhalt vor einem Element zu notieren. Durch die Selektoren `h1:before`, `h2:before` und `h3:before` wird für Überschriften 1., 2. und 3. Ordnung jeweils festgelegt, was vor einer solchen Überschrift automatisch eingefügt werden soll. Der Inhalt dessen, was einem Element automatisch vorangestellt werden soll, wird mithilfe der CSS-Eigenschaft `content` definiert. Das kann ein bestimmter, in Anführungszeichen zu setzender Text sein, wie z. B. `content:"Abschnitt: "`. In unserem Beispiel wird jedoch erst eine Funktion namens `counter()` notiert. Diese Funktion gibt den Wert einer Zählervariable aus. Bei h1-Überschriften hat die Zählervariable in unserem Beispiel den Namen `level_1`, bei h2-Überschriften `level_2` und bei h3-Überschriften `level_3`.

Der Wert, welcher `content:` zugewiesen wird, wird jedoch aus zwei oder mehreren Teilen zusammengesetzt. Die Teile werden jeweils durch Leerzeichen getrennt. Bei h1-Überschriften folgt hinter dem aktuellen Zählerstand der Zählervariable `level_1`, ermittelt durch die Funktion `counter()`, einfach ein Leerzeichen (`" "`). Es hat die Aufgabe, die Überschriftennummer vom Überschriftentext zu trennen.

Bei h2- und h3-Überschriften ist die Zusammensetzung des Werts für `content:` komplexer. Bei h2-Überschriften muss ja der aktuelle h1-Zähler mit ausgegeben werden, und bei h3-Überschriften der aktuelle h1-Zähler und der aktuelle h2-Zähler. Zwischen den Zahlen soll jeweils ein Punkt stehen. Auch dieser muss explizit angegeben werden (`"."`).

Nachdem eine Überschrift ihre korrekte Nummerierung erhalten hat, müssen jedoch noch die Zählervariablen verwaltet werden. Für diese eigentlich typischen Programmieranweisungen stellt CSS die Eigenschaften `counter-increment` (Zählervariable um 1 erhöhen) und `counter-reset` (Zähler auf 1 zurücksetzen) zur Verfügung.

Bei allen drei Überschriftentypen muss die jeweilige Zählervariable, nachdem eine Überschrift dieses Typs ihre aktuelle Nummerierung erhalten hat, um 1 erhöht werden. So erhöht `counter-increment:level_1` den Wert der Zählervariablen `level_1` für h1-Überschriften. Der Eigenschaft `counter-increment` wird also einfach die gewünschte Zählervariable zugewiesen.

Auf gleiche Weise funktioniert die Eigenschaft `counter-reset`. Die zugewiesene Zählervariable wird auf 1 zurückgesetzt. Dies ist wichtig, damit eine Überschriftenfolge wie h1 – h2 – h2 – h1 – h2 zu 1 – 1.1 – 1.2 – 2 – 2.1 führt und nicht zu 1 – 1.1 – 1.2 – 2 – 2.3.

6.2.2 Überschriftennummerierung mit JavaScript/DOM

Um die automatische Überschriftennummerierung auch für andere Browser zu ermöglichen, erstellen wir folgendes JavaScript in einer separaten Datei, die wir im Beispiel unter dem Namen ***enumerate.js*** abspeichern. Diese Datei erhält folgenden Inhalt:

Listing 6.7: Script mit Funktion enumerate() für Überschriftennummerierung von h1, h2, h3

```
function enumerate(h1_num) {
  if(!document.getElementsByTagName) return;
  if(window.opera) return;
  var body = document.getElementsByTagName("body")[0];
```

```
    h2_num = h3_num = 0;
    h1_num -= 1;
    for(i = 0; i < body.childNodes.length; i++) {
      if(body.childNodes[i].nodeName.toLowerCase() == 'h1') {
        h1_num += 1;
        h2_num = h3_num = 0;
        h1_text = body.childNodes[i].innerHTML;
        h1_numtext = h1_num + " " + h1_text;
        body.childNodes[i].innerHTML = h1_numtext;
      }
      else if(body.childNodes[i].nodeName.toLowerCase() == 'h2') {
        h2_num += 1;
        h3_num = 0;
        h2_text = body.childNodes[i].innerHTML;
        h2_numtext = h1_num + "." + h2_num + " " + h2_text;
        body.childNodes[i].innerHTML = h2_numtext;
      }
      else if(body.childNodes[i].nodeName.toLowerCase() == 'h3') {
        h3_num += 1;
        h3_text = body.childNodes[i].innerHTML;
        h3_numtext = h1_num + "." + h2_num + "." + h3_num + " " +
                     h3_text;
        body.childNodes[i].innerHTML = h3_numtext;
      }
    }
  }
```

Das Script besteht aus einer einzigen Funktion namens `enumerate()`. Auf Einzelheiten der Programmierung gehen wir in diesem Kapitel nicht näher ein. Wichtig zu wissen ist an dieser Stelle nur, was die Funktion leistet. So wie hier notiert, wertet sie alle h1-, h2- und h3-Elemente eines HTML-Dokuments aus und setzt ihrem Elementinhalt die Nummerierung voran. Dabei erwartet die Funktion beim Aufruf eine Zahl. Diese Zahl interpretiert sie als die Basiszahl für die erste h1-Überschrift im Dokument. Wenn Sie der Funktion also beispielsweise den Wert 4 übergeben, nummeriert sie 4, 4.1, 4.1.1 usw.

Die Funktion ist so geschrieben, dass Opera-Browser sie nicht bis zum Ende durchlaufen. Der Grund ist, dass Opera wie im vorherigen Abschnitt gesehen bereits die Nummerierung via CSS umsetzt. Falls Sie keine CSS-basierte Nummerierung verwenden möchten, sondern nur das hier vorgestellte JavaScript, dann müssen Sie in der Funktion folgende Zeile löschen:

```
if(window.opera) return;
```

Nun muss die Funktion noch im HTML-Dokument aufgerufen werden. Der vollständige Quelltext des Beispieldokuments lautet:

Listing 6.8: HTML-Dokument mit automatisch nummerierten Überschriften

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

```
    <head>
       <meta http-equiv="content-type" content="text/html;
             charset=ISO-8859-1">
       <title>Nummerierte Überschriften</title>
       <link rel="stylesheet" type="text/css"
             href="h-styles.css">
       <script type="text/javascript" src="enumerate.js">
       </script>
    </head>
    <body onLoad="enumerate(1)">
       <h1>Die Fauna (Tierwelt)</h1>
       <h2>Die Avifauna (Vögel)</h2>
       <h3>Laufvögel</h3>
       <h3>Hühnervögel</h3>
       <h3>Gänsevögel</h3>
       <h3>Kranichvögel</h3>
       <h2>Die Entomofauna (Insekten)</h2>
       <h3>Felsenspringer</h3>
       <h3>Fischchen</h3>
       <h3>Fluginsekten</h3>
       <h2>Die Ichthyofauna (Fische)</h2>
    </body>
</html>
```

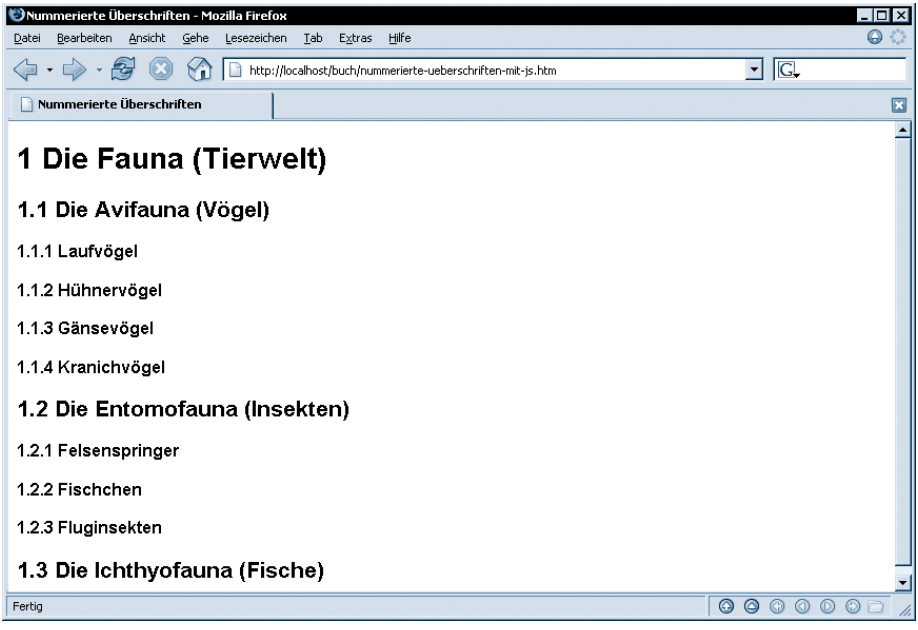

Abbildung 6.7: Automatische Überschriftennummerierung dank JavaScript

Im Kopf des HTML-Dokuments werden sowohl die CSS-Datei *h-styles.css* (zu deren Inhalt siehe Listing 6.6 ***auf Seite 293***) als auch die externe Scriptdatei *enumerate.js* eingebunden. Letztere enthält die weiter oben vorgestellte Funktion `enumerate()`. Eingebunden

wird das Script über ein script-Element, das mit Anfangs- und End-Tag notiert werden muss. Im src-Attribut wird die Scriptdatei als URI angegeben.

Damit ist das Script mit der Nummerierungsfunktion zwar eingebunden, aber die Nummerierungsfunktion muss noch explizit aufgerufen werden. Das darf aber erst geschehen, nachdem der Browser alle Überschriften des Dokuments eingelesen hat. Der Funktionsaufruf erfolgt deshalb im einleitenden <body>-Tag mithilfe des Event-Handlers `onLoad=`. Dieser Event wird ausgelöst, wenn der Browser das Dokument vollständig eingelesen hat. Die Überschriften werden zu diesem Zeitpunkt bereits angezeigt, und zwar ohne Nummerierung. Die Funktion `enumerate()` erledigt ihre Arbeit jedoch blitzschnell und in den meisten Fällen wird ein Anwender gar nicht merken, dass die Überschriften zunächst einen Moment lang nicht nummeriert dargestellt werden.

Die Funktion `enumerate()` wird im Beispiel mit dem Wert 1 aufgerufen. Das bedeutet, die Zählung für h1-Überschriften beginnt bei 1.

Die Nummerierung via JavaScript funktioniert nur, wenn ein Anwender JavaScript in seinem Browser aktiviert hat, und wenn der Browser DOM-fähiges JavaScript interpretiert. Damit keine JavaScript-Fehler auftreten, verhindert die Funktion `enumerate()`, dass unfähige Browser sie vollständig ausführen. Alle heute verbreiteten und webtauglichen Browser sollten die Funktion jedoch korrekt ausführen.

6.3 @-Regeln in CSS

Zur spezielleren Syntax von CSS zählen die »at-Regeln«. Folgende Regeln gehören dazu:

- die @import-Regel zum Einbinden anderer, in einer externen CSS-Datei notierten Formatdefinitionen,
- die @media-Regel zum Unterscheiden ausgabetypspezifischer Formatdefinitionen,
- die @page-Regel für druckorientierte Stylesheet-Definitionen,
- die @charset-Regel für CSS-eigene Zeichensatzdefinitionen.

Eine Regel kann eine einzelne Verarbeitungsanweisung sein, die mit einem abschließenden Semikolon endet. Es gibt jedoch auch Regeln, an die sich ein kompletter Block anschließt, begonnen und beendet durch geschweifte Klammern – genau so wie Formatdefinitionen zu einem Selektor.

6.3.1 Die @import-Regel

In den bisherigen Beispielen haben wir externe CSS-Dateien stets über `<link rel= "stylesheet" ...>` eingebunden. CSS bietet mit der @import-Regel jedoch auch eine eigene Möglichkeit zum Einbinden externer Stylesheets an. Ein Beispiel:

```
<style type="text/css">
   @import url(styles.css) screen;
   @import url(../special/styles-print.css) print;
```

```
    #example {
       background-color:yellow;
    }
</style>
```

Die @import-Regel wird innerhalb eines zentralen style-Bereichs im Kopf einer HTML-Datei notiert. Wichtig ist zunächst zu wissen, dass die @import-Regel vor allen anderen Definitionen im Stylesheet notiert werden muss.

Die Quelle der externen CSS-Datei wird über `url(...)` angegeben. Erwartet wird eine URI-Angabe. Die Datei kann also entweder lokal oder in Form einer absoluten Internetadresse referenziert werden.

Hinter der URL-Angabe kann sofort das abschließende Semikolon stehen. Es kann jedoch auch noch ein so genannter Medientyp angegeben werden. In diesem Fall wird angewiesen, dass die CSS-Definitionen aus der eingebundenen Datei nur für das angegebene Ausgabemedium gelten. Welche Ausgabemedien hier angegeben werden können, beschreiben wir weiter unten in Zusammenhang mit der @media-Regel.

6.3.2 Die @media-Regel

CSS erlaubt es, bei Formatdefinitionen nach Ausgabemedien zu unterscheiden, so beispielsweise die Unterscheidung nach Bildschirm- und Druckerausgabe. Ein Browser kann dann beim Anwenderwunsch, eine Seite auszudrucken, die Formate des print-Stylesheets verwenden, während für die Bildschirmanzeige die screen-Definitionen berücksichtigt werden. Gerade in Verbindung mit positionierten Elementen und mehreren Spalten kann es sinnvoll sein, für Ausdrucke ein eigenes Stylesheet zu entwickeln, welches die Elemente papierfreundlicher verteilt.

Daneben gibt es aber auch speziellere Medientypen, um der Vielfalt internetfähiger Geräte und auch Spezialausgabegeräten für Anwender mit Seh- oder anderen Schwächen gerecht zu werden. Die nachfolgende Tabelle listet die in der CSS-Spezifikation 2.1 vorgesehenen Medientypen auf. Spätere CSS-Versionen können weitere oder geänderte Medientypen enthalten. Maßgeblich dafür sind nicht zuletzt Entwicklungen auf dem Hardwaremarkt.

Medientyp	Bedeutung
`all`	Alle Medientypen (Default)
`braille`	Taktile Ausgabemedien mit Braille-Zeile
`embossed`	Seitenorientierte Braille-Drucker
`handheld`	Kleingeräte mit Display (Pocket-Computer, Handys, Tablet-PCs)
`print`	Drucker
`projection`	Beamer, Projektoren
`screen`	Übliche moderne farbige Computerbildschirme
`speech`	Sprach-Synthesizer (anstelle von `speech` ist auch `aural` möglich)

Tabelle 6.1: Medientypen in CSS

Medientyp	Bedeutung
tty	Textmodus-Ausgaben (Teletext, manche Handys, Textbrowser)
tv	Fernseher (eingeschränktes Scrollen, dafür soundfähig)

Tabelle 6.1: Medientypen in CSS (Fortsetzung)

Die @media-Regel wird als Block um die jeweils zugehörigen Formatdefinitionsblöcke gesetzt. Ein Beispiel:

```
@media print {
   body {
      background-color:white;
      color:black;
   }
   p,li,th,blockquote {
      font-family:Garamond, serif
      font-size:10pt;
   }
}
@media screen {
   body {
      background-color:black;
      color:white;
   }
   p,li,th,blockquote {
      font-family:Verdana, sans-serif;
      font-size: 10pt ;
   }
}
```

So wie im Beispiel notiert, könnte das Konstrukt in einer separaten CSS-Datei oder in einem style-Bereich innerhalb der HTML-Dateikopfdaten stehen.

Das Beispiel zeigt einige typische Formatdefinitionen, bei denen eine Unterscheidung zwischen Bildschirm (`@media print`) und Drucker (`@media print`) sinnvoll ist. Wenn bei der Bildschirmpräsentation helle Schriften auf dunklem Hintergrund verwendet werden, so ist es gut, für Druckausgaben dunkle Schriften auf weißem Grund anzugeben. Zwar verhindern die Browser zum Teil eigenmächtig, dass eine rabenschwarze Seite beim Ausdrucken eine halbe Tonerkartusche verbraucht, doch eine explizite Anweisung in CSS sorgt für Sicherheit. Auch Schriftarten und Schriftgrößen sind oft für die Bildschirmpräsentation optimiert. So ist die beliebte Microsoft-Schriftart Verdana vorwiegend für die Verwendung auf Websites entwickelt worden, weniger für Druckdokumente. Und Schriftgrößenangaben in Pixel sind ebenfalls bildschirmorientiert. Eine Unterscheidung nach Medientyp erlaubt es, für die Druckausgabe Punktangaben zur Schriftgröße zu notieren, was für die Bildschirmausgabe wiederum nicht empfehlenswert ist.

Beachten Sie bei der Syntax die verschachtelten Blöcke. Die @media-Regel schließt alles ein, was zwischen der öffnenden geschweiften Klammer hinter der Medientypangabe und ihrem Gegenstück, der schließenden geschweiften Klammer steht. Dazwischen befinden

sich typische Selektoren mit Formatdefinitionen, die ebenfalls wie üblich in geschweiften Klammern stehen, also eigene Blöcke darstellen.

Es ist auch erlaubt, mehrere Medientypen anzugeben, also z. B.:

```
@media screen, projection, tv { ... }
```

In diesem Fall gelten die zugeordneten Formatdefinitionen für alle genannten Gerätetypen.

6.3.3 Die @page-Regel

Das CSS-Boxmodell geht von einem *Viewport* aus. Bei der Ausgabe einer Webseite im Browser ist dies das aktuelle Anzeigefenster im Browser. Bei Ausgabe auf einen Drucker ist es – einen seitenorientierten Drucker, also keinen Endlosdrucker vorausgesetzt – eine Seite. Eine Seite kann jedoch unterschiedlich groß sein. So weichen beispielsweise das amerikanische US-Letter-Format und das vergleichbare deutsche A4-Format in ihren Längen- und Breitenverhältnissen voneinander ab. Ferner wird bei Druckseiten, die später gebunden werden sollen, in der Regel zwischen rechten und linken Seiten unterschieden. Für solche druckspezifischen Bedürfnisse steht die @page-Regel zur Verfügung.

Die Unterstützung der @page-Regel durch die Browser ist jedoch noch sehr dürftig. Die beste Unterstützung leistet derzeit der Opera-Browser.

Das nachfolgende Beispiel stellt einen möglichen Einsatz der @page-Regel vor:

```
@page {
   size:21.0cm 14.85cm;
   margin-top:2.2cm;
   margin-bottom:2.0cm;
}
@page :left {
   margin-left:1.7cm;
   margin-right:2cm
}
@page :right {
   margin-left:2cm;
   margin-right:1.7cm
}
```

Eine speziell für die @page-Regel geschaffene CSS-Eigenschaft ist `size`. Damit lässt sich innerhalb des Blocks einer @page-Regel die gewünschte Seitengröße definieren. Als Wert werden entweder zwei numerische Angaben erwartet, von denen die erste als Breite und die zweite als Höhe der Seite interpretiert wird. Oder Sie geben nur das allgemeine Seitenformat vor, nämlich `portrait` (Hochformat) oder `landscape` (Querformat). Bei numerischen Angaben sind printorientierte Einheiten wie `cm`, `mm` oder `inch` dringend zu empfehlen.

Über die Adressierung `@page:left` und `@page:right` können Sie zwischen linken und rechten Seiten beim Druck unterscheiden. Im obigen Beispiel werden die linken und rechten

Ränder entgegengesetzt definiert, was beim Ausdruck der Berücksichtigung eines Binderands (im Beispiel 0.3 Zentimeter) entspricht.

Weitere druckspezifische CSS-Formatierungen

Die folgenden CSS-Eigenschaften gehören nicht zur @page-Regel, sind jedoch in Zusammenhang mit printorientierter Ausgabe von Bedeutung. Vor allem innerhalb von Formatdefinitionen für den Medientyp `print` können solche Angaben sinnvoll sein. Die Eigenschaften können innerhalb aller üblichen Selektoren notiert werden.

Drei Eigenschaften namens `page-break-before`, `page-break-after` und `page-break-inside` erzwingen und verhindern Seitenumbrüche. Ein Beispiel:

```
@media print {
   h1 {
      font-family:Tahoma,sans-serif;
      font-size:15pt;
      page-break-before:always;
   }
   p, li, blockquote {
      font-family:inherit;
      font-size:10pt;
      page-break-inside:avoid;
   }
}
```

Die Eigenschaften `page-break-before` (Seitenumbruch vor diesem Element) und `page-break-after` (Seitenumbruch nach diesem Element) erlauben neben den Standard-Wertzuweisungen `auto` und `inherit` (Voreinstellung und »wie Elternelement«) die Angaben `always` (Seitenumbruch in jedem Fall erzwingen), `left` (Seitenumbruch erzwingen, und zwar so, dass die nächste Seite eine linke Seite ist), `right` (Seitenumbruch erzwingen, und zwar so, dass die nächste Seite eine rechte Seite ist) sowie `avoid` (Seitenumbruch verhindern). Bei `left` und `right` kann es vorkommen, dass zwei Seitenvorschübe eingefügt werden und eine Leerseite entsteht. Bei `page-break-inside` (Seitenumbruch innerhalb dieses Elements) ist neben den Standardzuweisungen nur `avoid` (Seitenumbruch innerhalb dieses Elements verhindern) erlaubt. Der Elementinhalt wird dann komplett auf die nächste Seite gedruckt. Auf der Seite davor bleibt unten je nach Umfang ein gewisser Leerraum.

Ebenfalls für seitenorientierte Ausgabemedien gedacht sind die beiden Eigenschaften `orphans` (alleinstehende Zeilen am Seitenende) und `widows` (alleinstehende Zeilen am Seitenanfang). Ein Beispiel:

```
@media print {
   p, li, blockquote {
      font-family:inherit;
      font-size:10pt;
      orphans:2;
      widows:2;
   }
}
```

In diesem Beispiel wird festgelegt, dass Fließtext der HTML-Elemente `p`, `li` und `blockquote`, falls innerhalb davon beim Ausdruck ein Seitenwechsel vorkommt, so umgebrochen werden sollte, dass wenigstens zwei Textzeilen am Ende der Seite und zwei Zeilen auf der neuen Seite stehen. Ist der Text kürzer, wird er komplett auf die neue Seite gedruckt. Der Wert für beide Eigenschaften ist also eine einfache Zahl, welche für die Anzahl Textzeilen steht.

6.3.4 Die @charset-Regel

Die @charset-Regel hat nichts mit dem HTML-Zeichensatz zu tun. Innerhalb von HTML-Dokumenten sollte die Angabe zum Zeichensatz deshalb stets wie üblich über die dafür vorgesehene Meta-Angabe erfolgen.

Die @charset-Regel ist nur für externe CSS-Dateien gedacht, damit ein auslesender Browser weiß, nach welchem Zeichensatz er die CSS-Datei dekodieren muss. Die @charset-Regel muss ganz zu Beginn der CSS-Datei notiert werden, z. B.:

```
@charset "UTF-8";
```

Innerhalb der Anführungszeichen muss ein Zeichenvorratsname aus dem IANA-Repertoire angegeben werden, also entweder eine Zeichenkodierungsform wie UTF-8 oder ein Zeichensatz wie beispielsweise ISO-8859-1. Die CSS-Datei muss im entsprechenden Kodierungsformat bzw. Zeichensatz abgespeichert sein.

6.4 CSS und die Browser

In Kapitel 5 haben Sie gelernt, wie moderne Webseitenlayouts ausschließlich über CSS erstellt werden. Nun gibt es ältere Browser, die vereinzelt noch zum Einsatz kommen, welche aber modernes CSS nicht so interpretieren wie spezifiziert oder nur einen Teil davon. Gemeint sind vor allem Netscape-Browser der 4er-Generation, Internet Explorer der 4er-Generation, teilweise auch noch der 5er-Generation, und Opera-Browser unter Version 6. Manchmal ist es sinnvoll, diese Browser mithilfe kleiner Tricks auszuschließen oder explizit anzusprechen. Diese Technik der so genannten ***Browser-Weichen*** versucht, Code bedingt ausführbar oder interpretierbar zu machen. Allein innerhalb von CSS gibt es einige Möglichkeiten dafür. In diesem Abschnitt werden wir einige wichtige davon vorstellen.

Schuld an der Misere sind übrigens nicht nur die (bösen, kommerziellen) Browser-Anbieter, wie von Hardlinern gerne behauptet wird. Auch die CSS-Spezifikation des W3-Konsortiums ist ein historisch gewachsenes Gebilde und war in früheren Jahren bei weitem nicht so präzise, wie sie es mittlerweile ist. Beschreibungen wie etwa das Boxmodell fehlten zunächst einfach. Der Interpretationsspielraum, den die Spezifikation ließ, führte bei den Browsern zu unterschiedlichen Implementierungen.

6.4.1 Netscape 4.x ausschließen

Wenn Sie überhaupt noch Wert darauf legen, auf Netscape-4-Browser Rücksicht zu nehmen, dann sollten Sie, falls Sie CSS massiv und vor allem als Layoutbasis einsetzen, diesen Browser völlig ausschließen und dafür sorgen, dass er nur nacktes HTML anzeigt, ebenso wie ein textbasierter Browser.

Dies geht recht einfach mithilfe der @import-Regel. Notieren Sie alle CSS-Formatdefinitionen in einer externen CSS-Datei. Binden Sie diese Datei in HTML nicht über das `<link>`-Tag ein (diese Syntax versteht Netscape 4 nämlich), sondern CSS-spezifisch (das ignoriert Netscape 4). Beispiel:

```
<style type="text/css">
   @import url(styles.css);
</style>
```

Diese Lösung ist in jedem Fall sauberer, als Netscape mit CSS-Definitionen zu konfrontieren, die er völlig fehlerhaft und teilweise mit grotesken Bugs darstellt. Vorausgesetzt, Ihre HTML-Dokumente enthalten ordentlich strukturiertes Markup, steht einer akzeptablen Präsentation in Netscape nichts im Wege.

Netscape-4-Browser mit eigenen Formaten bedienen

Wenn Sie sich die Mühe machen wollen, können Sie Netscape 4 doch noch mit etwas für ihn verdaulichem CSS versorgen. Binden Sie dazu zusätzlich zur Referenz via @import-Regel über das <link>-Tag eine andere CSS-Datei ein, die nur jene Formate enthält, die auch für Netscape unkritisch sind. Beispiel:

```
<link rel="stylesheet" type="text/css" href="simple.css">
<style type="text/css">
   @import url("advanced.css");
</style>
```

Mit diesem Code werden zwei CSS-Dateien eingebunden. Die Datei *simple.css* wird von allen Browsern interpretiert, die Datei *advanced.css* jedoch nicht von Netscape-4-Browsern. In der *simple.css* könnten beispielsweise Angaben zur Schriftformatierung stehen, die Netscape 4 noch leidlich umsetzt. Formatdefinitionen, die eine ordentliche Umsetzung des CSS-Boxmodells erfordern, wie Rahmen, Hintergrundgestaltung, Positionierung, Breiten, Höhen usw., sollten Netscape 4 dagegen unzugänglich bleiben und könnten in der *advanced.css* stehen.

6.4.2 Internet Explorer ausschließen und explizit ansprechen

Zurzeit der Drucklegung dieses Buchs ist der Internet Explorer der in der Szene am meisten verpönte Browser, leider aber auch noch der mit der größten Verbreitung. Verpönt ist er nicht nur wegen seiner engen Betriebssystemeinbindung, die ihn zu einem großen Sicherheitsrisiko für PCs werden ließ, sondern auch, weil er bei der Umsetzung der aktuellen HTML- und CSS-Standards des W3-Konsortiums noch einige ärgerliche Lücken und Fehler aufweist.

Wichtig ist auf jeden Fall, im HTML-Dokument eine gültige und vollständige Dokumenttyp-Deklaration zu notieren (inklusive DTD-URL-Angabe). Nur dadurch wird der Internet Explorer 6 dazu bewegt, vom so genannten *Quirks Mode* in den so genannten *Compliant Mode* umzuschalten. Im Quirks Mode benutzt der Internet Explorer ein älteres, microsoft-eigenes CSS-Boxmodell, das gewisse Abweichungen im Vergleich zu der CSS-Spezifikation aufweist. Die Ausdehnungssummen von Elementinhalt, Innenabstand, Rahmen und Außenabstand werden in diesem Modell anders berechnet und können daher bei Seitenlayouts zu unangenehmen Überraschungen führen. Der Compliant Mode ist dagegen der Schalter für eine standardkonforme Interpretation. Gesetzt wird der Schalter wie erwähnt, indem in HTML eine vollständige Angabe zum Dokumenttyp notiert wird.

Internet Explorer ausschließen

Einige neuere syntaktische Features bei Selektoren versteht der Internet Explorer bis einschließlich Version 6.0 nicht. Formatdefinitionen innerhalb von Selektoren mit solchen syntaktischen Elementen ignoriert er vollständig. Wenn Sie also Formate definieren möchten, die der Internet Explorer nicht interpretieren soll, dann müssen Sie einfach den Selektor so formulieren, dass dieser Browser ihn ignoriert. Ein Beispiel:

Selektor für alle Browser:

```
#kopf {
   position:fixed;
   top:0;
   left:0;
   width:100%;
   height:80px;
}
```

Selektor nicht für Internet Explorer:

```
body > #kopf {
   position:fixed;
   top:0;
   left:0;
   width:100%;
   height:80px;
}
```

Die #-Syntax, um via Selektor ein Element mit einem bestimmten id-Namen auszuwählen, kennt der Internet Explorer. Die Syntax, um mithilfe des >-Zeichens eine genaue Angabe zur Verschachtelungsregel zu notieren, kennt er hingegen nicht und er ignoriert sämtliche zugehörigen Formatdefinitionen. Befindet sich das Element mit dem id-Namen `#kopf` in der Elementhierarchie direkt unterhalb des body-Elements, also `<body> ... <element> </element> ... </body>`, so ist die zweitgenannte Selektor-»Formulierung« bedeutungsgleich mit der ersten. Im Gegensatz zur ersten wird sie jedoch vom Internet Explorer ignoriert, was sinnvoll ist, da dieser `position:fixed` nicht kennt.

Auch Selektoren mit attributbedingten Angaben wie `div[id]` schließen den Internet Explorer aus.

Eine weitere Möglichkeit, Internet Explorer mit einer Versionsnummer kleiner als 5 auszuschließen, besteht darin, alle zu verbergenden Formatdefinitionen in einen @media-Block zu packen. Ein Beispiel:

```
@media all {
   p:first-letter { font-size:250%; }
}
```

Da der Internet Explorer die @-Regeln bis Version 4 noch nicht kennt, ignoriert er sie. Die Angabe `@media all` (also »alle Medien«) hat für andere Browser jedoch den gleichen Effekt, wie wenn der @media-Block gar nicht da wäre.

Mit den beschriebenen Maßnahmen wird gleichzeitig auch Netscape 4.x ausgeschlossen.

Internet Explorer explizit ansprechen

Die zweifellos eleganteste Methode, Regelungen speziell für den Internet Explorer zu treffen, sind die genannten ***Conditional Comments***. Dies sind HTML-Kommentare, deren Inhalt vom Internet Explorer auf Grund einer speziellen Syntax jedoch nicht ignoriert, sondern interpretiert wird. HTML-Kommentare beginnen normalerweise mit `<!--` und enden mit `-->`. Mit `<!--[if ...]>` und dem Gegenstück `<![endif]-->` können Sie jedoch HTML-Code speziell für den Internet Explorer oder sogar für spezielle Versionen davon notieren. Andere Browser behandeln den Inhalt dagegen als normalen HTML-Kommentar und ignorieren ihn. Die Syntax zur Formulierung der if-Bedingung erlaubt Ausprägungen wie `[if IE]` (wenn es irgendein Internet Explorer ist), `[if IE 6]` (wenn es ein Internet Explorer 6.0 ist), `[if gte IE 5]` (wenn es ein Internet Explorer Version größer gleich 5.0 ist) oder `[if lte IE 6]` (wenn es ein Internet Explorer mit einer Version unter 6.0 ist).

In Zusammenhang mit CSS erlauben es solche Kommentare, `<link>`-Tags zum Einbinden spezieller CSS-Dateien oder spezifische style-Bereiche zu notieren. Ein Beispiel:

```
<!--[if lte IE 6]>
   <link rel="stylesheet" type="text/css" href="ie-lte-6.css">
   <style type="text/css">
      div.kasten {
         width:60%;
         height:100px;
         background-color:yellow;
      }
   </style>
<![endif]-->
```

In diesem Beispiel wird sowohl eine CSS-Datei eingebunden als auch ein Style-Bereich definiert. Der Code steht jedoch innerhalb der Zeichenfolgen `<!-- -->` und wird deshalb eigentlich von allen Browsern ignoriert. Die spezielle Anschlusssyntax signalisiert dem

Internet Explorer jedoch, dass er den HTML-Code interpretieren soll, wenn es sich um eine Produktversion kleiner als 6.0 handelt.

Links zum Thema Browser-Weichen mit CSS:

Browser-Weichen und -Filter per CSS:
http://www.lipfert-malik.de/webdesign/tutorial/bsp/css-weiche-filter.html
Workshop Browser-Weiche:
http://www.css4you.de/wsbw/