Skriptum

HTML

1. und 2. Jahrgänge
Betriebsinformatik

TGM,
Abt. Wirtschaftsingenieurwesen

HTML

Einleitung

HTML, vom Web-Gründer Tim Berners-Lee entwickelt, wurde im Zuge des Web-Booms zum erfolgreichsten und meist verbreiteten Dateiformat der Welt. Immer wieder rümpfen Entwickler, die gerne alles komplizierter machen, um ihren technischen Durst zu stillen, über HTML die Nase - eben weil es so einfach ist. Aber erstens ist es bei genauerem Hinsehen gar nicht so einfach und zweitens reicht HTML für die Mehrzahl der Inhalte, die heute im Web angeboten werden, vollkommen aus. Denn HTML ist eine Sprache zur Strukturierung von Texten, wobei aber auch die Möglichkeit besteht, Grafiken und multimediale Inhalte in Form einer Referenz einzubinden und in den Text zu integrieren.

Mit HTML kann man Überschriften, Textabsätze, Listen und Tabellen erzeugen. Man kann anklickbare Verweise auf beliebige andere Web-Seiten oder Datenquellen im Internet erzeugen. Nicht-textuelle Inhalte können wie bereits erwähnt referenziert werden. Man kann Formulare in den Text integrieren. Und last but not least bietet HTML Schnittstellen für Erweiterungssprachen wie CSS-Stylesheets oder JavaScript an, mit deren Hilfe man HTML-Elemente nach Wunsch gestalten und formatieren oder Interaktion mit dem Anwender realisieren kann.

HTML ist also sehr durchdacht und in ein Gesamtkonzept an weiteren Sprachen eingebunden. Wer behauptet, HTML sei nichts mehr für professionelles Web-Publishing, outet sich meistens nur als Technik-Junkie und behauptet letztlich nur, normale Texte seien nichts mehr für professionelles Web-Publishing. Und wer das behauptet, stellt sich selber ins Abseits der gewachsenen schriftlichen Kultur. Solange es Autoren gibt, die Texte schreiben und dazu die üblichen Mittel zur Textstrukturierung benötigen, wird es aus web-technischer Sicht Leute geben, die HTML gut brauchen können.

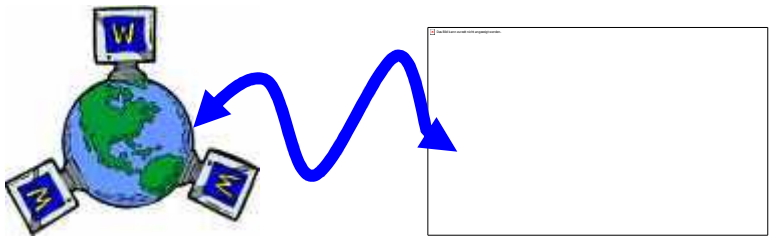
HTML ist allerdings mit diversen spezielleren Anforderungen schlichtweg überfordert. HTML allein kann weder Grafik-Designer befriedigen, die jeden Pixel am Bildschirm kontrollieren wollen, noch Daten-Designer, die aus der Welt der relationalen Datenbanken kommen und sich anwendungs-spezifische Lösungen wünschen. Deshalb gibt es heute Style-Sprachen wie CSS, und es gibt Lösungen wie XML, um anwendungs-spezifisches Daten-Design zu ermöglichen.

All das ändert aber nichts daran, dass HTML eine hervorragend geeignete, standardisierte und wegen der weiten Verbreitung der Web-Browser praktisch überall verfügbare Sprache für Text und Hypertext darstellt. Das W3-Konsortium, das für die Standardisierung von HTML zuständig ist, ist zwar bemüht, HTML von allen Sünden der Anfangsjahre zu reinigen und es als einfache, reine Text-Strukturierungssprache zu etablieren. Doch in der Praxis dient HTML heute auch als Basis zum Erstellen von Web-Seiten-Layouts, und daran wird sich wohl auch so schnell nichts ändern.

Kleines Lexikon

HTML steht für „Hypertext Markup Language“ und ist das Format, in dem die Text- und Hypertext-Informationen im WWW (World Wide Web) gespeichert und übertragen werden.

Unter **Hypertext** versteht man Texte mit Querverweisen, die ähnlich wie in einem Lexikon oder in einer Literaturliste die Verbindung zu weiteren Informationen herstellen.
Im WWW werden solche Verweise mit Hilfe von **URL**s (Uniform Resource Locator) realisiert.
URL ist die „Adresse“, die das Client-Programm benötigt, um eine bestimmte Information vom jeweiligen Server-Computer zu erhalten. Der URL enthält zu diesem Zweck Informationen wie die Art des Zugriffs (Protokoll), die Adresse des Server-Computers (Hostname), eventuell mit einem Username und Passwort oder einer Port-Nummer, und das Directory und den Filenamen der Datei, in der die gewünschte Information gespeichert ist.
Das **WWW** ist ein Informationssystem, das einen Zugriff auf Informationen, die auf vielen verschiedenen Computern gespeichert sind ermöglicht. Der Zugriff erfolgt nach dem Prinzip von Client und Server über das Internet mit dem Protokoll HTTP (Hypertext Transfer Protocol).
Server sind Computer, auf denen die Informationen gespeichert sind. **Clients** sind die Benutzer, die Informationen haben wollen. Client-Programme sind Programme, mit denen die Benutzer von ihren eigenen Rechnern (PCs) aus auf die Informationen, die auf den Servern gespeichert sind, zugreifen. WWW-Client-Programme werden auch als Web-Browser bezeichnet.



Web-Browser laufen meist auf einer grafischen Benutzeroberfläche mit Maus oder Toch-Screen. Spezielle Browser-Programme können die Informationen auch in Zeilen-orientierten oder in Blindenschrift oder akustisch (als gesprochener Text) ausgeben. Typische Web-Browser waren bzw. sind Mosaic, Lynx, Netscape, Internet-Explorer, Opera, WebTV, iPlanet, Mozilla. Die meisten Web-Browser unterstützt nicht nur den Zugriff auf WWW-Server sondern auch auf andere Internet-Service wie Telnet, GTP, Electronic Mail uns Usenet News.
HTTP ist das Protokoll, nach dem die Informationen zwischen WWW-Servern und WWW-Clients über das Internet übertragen werden. Es gibt auch ein „Secure HTTP“.
Secure HTTP ermöglicht die abhör- und fälschungssichere Übertragung der Informationen zwischen Web-Browser und Web-Server.

Grundgerüst einer HTML-Datei

Eine gewöhnliche HTML-Datei besteht grundsätzlich aus folgenden Teilen:

- Dokumenttyp-Angabe (Angabe zur verwendeten HTML-Version)
- Header (Kopfdaten. z.B. Angaben zu Titel u.ä.)
- Body (Körper - anzuzeigender Inhalt, also Text mit Überschriften, Verweisen, Grafikreferenzen usw.)

Schema

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
       "http://www.w3.org/TR/html4/transitional.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

</body>
</html>
```

Erläuterung

Die erste Zeile sieht für Anfänger zunächst verwirrend aus. Diese etwas komplizierte Angabe ist eine Dokumenttyp-Angabe.

Dokumenttyp

HTML ist innerhalb der Familie der Auszeichnungssprachen nur eine von vielen, wenn auch die prominenteste. HTML selbst hat außerdem bereits eine mehrjährige Geschichte und ist in verschiedenen, recht unterschiedlichen Versionen normiert worden. Mit der Dokumenttyp-Angabe bestimmt man, welche Auszeichnungssprache in welcher Version verwendet wird. Eine auslesende Software, etwa ein Web-Browser, kann sich an dieser Angabe orientieren.

Nach den Regeln einer Auszeichnungssprache ist eine HTML-Datei oder eine XHTML-Datei erst dann eine **gültige** (**valide**) Datei, wenn sie einen bestimmten Dokumenttyp angibt und sich dann innerhalb des restlichen Quelltextes genau an die Regeln hält, die in diesem Dokumenttyp definiert sind. Denn hinter jeder Dokumenttyp-Angabe stecken so genannte **Dokumenttyp-Definitionen** (**DTD**). Auch für HTML gibt es solche Dokumenttyp-Definitionen. Dort ist geregelt, welche Elemente ein Dokument vom Typ HTML enthalten darf, welche Elemente innerhalb von welchen anderen vorkommen dürfen, welche Attribute zu einem Element gehören, ob die Angabe dieser Attribute Pflicht ist oder freiwillig usw.

Als HTML-Anfänger mag dir der Aufwand, der da betrieben wird, vielleicht nicht ganz einleuchten. Doch genau diese Dokumenttypen, mit deren Hilfe sich Regeln für Sprachen wie HTML genau definieren lassen, sind ein riesiger Fortschritt in der EDV. Denn nur so lässt sich das Konzept der software-unabhängigen, aber regelgerechten Dateiformate konsequent durchsetzen. Ohne den Bezug auf die offiziellen Regeln wären Sprachen wie HTML unverbindliche Konventionen, die schnell in Dialekten verwässern würden. Das ist genauso wie bei natürlichen Sprachen: ohne eine gewisse Regelung der "Rechtschreibung" würde eine Schriftsprache im Laufe der Zeit so stark zersetzt, dass am Ende kaum jemand mehr lesen kann, was der andere mit dem was er geschrieben hat meint. Da Software außerdem noch viel dümmer ist als Menschen und viel genauere Vorgaben benötigt, um zu "verstehen" was man ihr mitteilt, ist das Beziehen auf Regeln dort sogar noch wesentlich wichtiger.

Verwendete Tags

Der gesamte übrige Inhalt einer HTML-Datei wird in die Tags `<html>` bzw. `</html>` eingeschlossen. Das `html`-Element wird auch als **Wurzelelement** einer HTML-Datei bezeichnet. Hinter dem einleitenden HTML-Tag folgt das einleitende Tag für den Kopf

<head>. Zwischen diesem Tag und seinem Gegenstück </head> werden Kopfdaten einer HTML-Datei notiert. Die wichtigste dieser Angaben ist der Titel der HTML-Datei, markiert durch <title> bzw. </title>. Unterhalb davon folgt der Textkörper, markiert durch <body> bzw. </body>. Dazwischen wird dann der eigentliche Inhalt der Datei notiert, also das, was im Anzeigefenster des WWW-Browsers angezeigt werden soll.

Wenn du Frames (mehrere Bildschirmfenster) einsetzen willst, sieht das Grundgerüst von Dateien, in denen ein Frameset definiert wird, anders aus. Dies wird in einem späteren Abschnitt beschrieben.

Erstellung einer HTML-Datei

Im Folgenden wird genau erklärt wie du deine erste HTML-Datei erstellen und anzeigen kannst:

- Öffne einen Texteditor. Ein Texteditor ist ein einfaches Programm mit dem man Textdateien erstellen kann, wie z.B.: „Editor“ unter Windows (ist unter Programme, Zubehör zu finden).
- Schreibe den angegebenen Code in den Editor.
- Speichere die Datei unter dem Namen „Beispiel1.html“ ab. (im Editor unter Windows stelle als Dateityp „Alle Dateien“ statt „Textdatei“ ein! Andernfalls wird die Datei als „Beispiel1.html.txt“ abgespeichert und der Browser erkennt die Datei nicht als HTML-Datei.)
- Öffne mit einem Doppelklick das File „Beispiel1.html“. Es öffnet sich dein Standardbrowser und zeigt den Text an.

Wenn du Änderungen an der Datei „Beispiel1.html“ vornehmen möchtest, so öffne den Editor. Im Editor öffne die Datei „Beispiel1.html“ (unter Windows vergiss nicht zuvor den Dateityp auf „alle Dateien“ zu ändern, ansonsten werden nur Textdateien angezeigt.)

Wichtig:

Willst du unter Windows die Datei „Beispiel1.html“ direkt im Verzeichnis ändern, so musst du mit der rechten Maustaste auf die Datei klicken und „Öffnen mit…“ anklicken, um das Programm „Editor“ auswählen zu können. Bitte achte darauf, dass der Haken bei „Diesen Dateityp immer mit diesem Programm öffnen“ **nicht** gesetzt ist. Bleibt der Haken gesetzt, so werden in Zukunft alle HTML-Dateien mit dem Editor anstatt mit dem Browser geöffnet!

Regeln zur Erstellung einer HTML-Datei

Zur Erstellung einer HTML-Dateie solltest du folgende Regeln kennen und beachten:

- Notiere in einer neuen Datei immer zuerst das (oben angegebene) Grundgerüst einer HTML-Datei.
- Beachte bei der Texteingabe die Vorschriften für Zeichen, Sonderzeichen und HTML-eigene Zeichen. (siehe unten)

- Setze Zeilenumbrüche und Leerzeilen so, dass du im Quelltext eine optimale Übersicht behältst. Beachte aber auch, dass Zeilenumbrüche und Absatzschaltungen im Web-Browser nicht so angezeigt werden, wie sie im Quelltext eingegeben wurden. Für Zeilenumbrüche und Absatzschaltungen, die im WWW-Browser wirksam sein sollen, musst du die entsprechenden HTML-Elemente verwenden, zum Beispiel diejenigen für Textabsätze oder Zeilenumbrüche. Wenn du aus besonderen Gründen Text im Web-Browser so anzeigen willst wie du ihn eingeben hast (mit allen Einrückungen, Umbrüchen usw.), kannst du das HTML-Element für präformatierten Text verwenden (`<pre>` und `</pre>`).
- Beachte, dass es in HTML keine Tabulatoren gibt. Ein im Quelltext eingegebener Tabulator wird bei der Anzeige im Web-Browser in ein Leerzeichen umgewandelt. Zeilenumbruchzeichen, Tabulatorzeichen und einfache Leerzeichen bilden in HTML die Klasse der so genannten **White-Space-Zeichen** (*white space = "weißer Raum"*). Die Browser setzen in der Regel ein Tabulatorzeichen oder Zeilenumbruchzeichen im Editor als Leerzeichen im HTML-Text um. Mehrere solcher White-Space-Zeichen hintereinander werden ignoriert und zu einem einzigen Leerzeichen zusammengefasst. Um mehrere Leerzeichen hintereinander zu erzwingen, kannst du anstelle der normalen Leerzeicheneingabe die Zeichenfolge   (geschütztes Leerzeichen) eingeben, und zwar so oft hintereinander wie gewünscht.

Zeichen, Sonderzeichen und HTML-eigene Zeichen

Wenn du Texte - sagen wir in deutscher Sprache - einfach in einen HTML-Editor eintippst, den Text mit Hilfe von HTML-Elementen strukturierst und dir das Ganze dann im Web-Browser anzeigen lässt, so wird in der Regel der gesamte eingegebene Text korrekt angezeigt. Das klingt selbstverständlich - ist es aber nicht. In der HTML-Datei stehen nämlich nicht deine eingegebenen Buchstaben und Satzzeichen, sondern Byte für Byte numerische Werte wie 75, 168 oder 32. Der Browser versucht nun herauszubekommen, nach welchem **Zeichensatz** er diese numerischen Werte interpretieren soll. Ob beispielsweise nach einem westeuropäischen, einem kyrillischen oder einem arabischen Zeichensatz. Wenn du keinerlei Angaben zum verwendeten Zeichensatz machst, dann wird der Browser am Ende seiner Bemühungen einfach den Zeichensatz verwenden, der in seinen Einstellungen voreingestellt ist. In deinem Browser, der vermutlich eine englische oder deutschsprachige Benutzerführung hat, ist vermutlich der Zeichensatz für westeuropäische Sprachen eingestellt, der so genannte **Latin-1**-Zeichensatz (ISO 8859-1). Weil der HTML-Editor oder Texteditor, den du beim Eintippen benutzt, vermutlich ebenfalls nach diesem Zeichensatz abspeichert, klappt alles wunderbar. Nun könnte es aber, wenn deine Web-Seiten im Web stehen, auch mal passieren, dass Besucher aus Osteuropa, Asien usw. vorbeikommen, die ganz andere Zeichensätze in ihren Web-Browsern voreingestellt haben. Solche Besucher werden dann lauter Zeichen ihres eigenen, vertrauten Zeichensatzes sehen - aber es wird ein wilder Zeichensalat sein, dem man beim besten Willen keinen Sinn entnehmen kann.

HTML bietet deshalb die Möglichkeit an, dem Browser mitzuteilen, welchen Zeichensatz du meinst. Dann liegt es am Browser, die von dir gemeinten Zeichen am Bildschirm beispielsweise eines Seitenbesuchers aus Fernost so anzuzeigen wie du sie eingetippt hast. Es gibt folgende Möglichkeiten, dem Browser mitzuteilen, welchen Zeichensatz

bzw. welches bestimmte Zeichen aus einem anderen als dem voreingestellten Zeichensatz du meinst:

- **global für eine Datei:** dazu gibt es die Möglichkeit, im Dateikopf einer HTML-Datei in einem so genannten Meta-Tag eine Angabe zum Default Zeichensatz zu notieren. Eine solche Angabe ist sehr zu empfehlen, da du dem Browser damit auf jeden Fall die Entscheidung leichter machst, nach welchem Zeichensatz die Bytes der HTML-Datei zu interpretieren sind:

- `<head>`

- `<meta http-equiv="content-type" content="text/html;`

- `      charset=ISO-8859-1">`

- `<!-- ... andere Angaben im Dateikopf ... -->`

- `</head>`

- `ISO-8859-1` ist der normale Zeichensatz für westeuropäische Sprachen, unter anderem auch für Deutsch.
- **für einzelne Zeichen:** Das ist vor allem dann sinnvoll, wenn du eine globale Angabe zum Zeichensatz gemacht hast, im Text aber einzelne Zeichen verwenden willst, die in dem angegebenen Zeichensatz nicht vorkommen. Dabei gibt es wiederum zwei Möglichkeiten: entweder du verwendest eine spezielle numerische Notation. Für häufiger verwendete Sonderzeichen stellt HTML aber auch so genannte **benannte Zeichen** zur Verfügung.

- Ersetze das Zeichen ä durch die Zeichenfolge `&auml;`

- Ersetze das Zeichen Ä durch die Zeichenfolge `&Auml;`

- Ersetze das Zeichen ö durch die Zeichenfolge `&ouml;`

- Ersetze das Zeichen Ö durch die Zeichenfolge `&Ouml;`

- Ersetze das Zeichen ü durch die Zeichenfolge `&uuml;`

- Ersetze das Zeichen Ü durch die Zeichenfolge `&Uuml;`

- Ersetze das Zeichen ß durch die Zeichenfolge `&szlig;`

- Beispiel

- `In M&uuml;nchen steht ein Hofbr&auml;uhaus.`

- `Dort gibt es Bier aus Ma&szlig;kr&uuml;gen.`

- Ersetze die deutschen Sonderzeichen wie im obigen Beispiel durch die entsprechenden Zeichenfolgen - keine Sorge, die Web-Browser verstehen das und zeigen die Zeichen korrekt an.

- Auch für das Eurozeichen gibt es ein **benanntes Zeichen** in HTML.
- `Preis: &euro; 199,-`
- Weitere benannte Zeichen findest du im Anhang.

Vermeide unter Windows, das Eurozeichen über die Tastatur erzeugt in HTML einzutippen ([AltGr]+[e]). Der Grund ist, dass Microsoft das Eurozeichen intern auf den Zeichenwert 128 gelegt hat, um es über Tastatur zugänglich zu machen und in vorhandene Schriftarten einzubauen. Das entspricht jedoch nicht dem Unicode/ISO-10646-Standard, auf dem HTML aufsetzt. Verwende in HTML deshalb das oben beschriebene `&euro` oder als Alternative eine numerische Notation nach dem Unicode/ISO-10646-Standard. Dort hat das Eurozeichen den Hexadezimalwert 20AC oder den Dezimalwert 8364. Nach HTML 4.0 kannst du das Eurozeichen demnach numerisch so referenzieren: `&#x20AC;` oder `&#8364;`.

HTML-eigene Zeichen maskieren

Wenn in deinem Text Zeichen vorkommen, die in HTML eine bestimmte Bedeutung haben, musst du diese Zeichen maskieren. Die folgenden Zeichen musst du wie folgt maskieren:

Ersetze das Zeichen < durch die Zeichenfolge `<`
Ersetze das Zeichen > durch die Zeichenfolge `>`
Ersetze das Zeichen & durch die Zeichenfolge `&`
Ersetze das Zeichen " durch die Zeichenfolge `&quot;`

Beispiel

```
Das ist ein &lt;HTML-Tag&gt;
GmbH &amp; Co. KG
&quot;Text steht in Anf&uuml;hrungszeichen&quot;
```

Am gefährlichsten ist die spitze öffnende Klammer (<). Wenn du dieses Zeichen nicht wie vorgeschrieben maskierst, bringst du den Web-Browser mit ziemlicher Sicherheit durcheinander, weil er glaubt, nun würde ein HTML-Tag folgen. Die anderen drei zu maskierenden Zeichen führen zwar meistens nicht zu Anzeigefehlern, doch solltest du sie auch stets maskieren. Besonders bei normalen Anführungszeichen, die ja doch sehr oft zum Einsatz kommen, wird die Maskierung im Text oft vergessen.

Elemente und Tags in HTML

HTML-Dateien bestehen aus Text. Zur Textauszeichnung gibt es bestimmte Zeichen aus dem normalen Zeichenvorrat.

Der Inhalt von HTML-Dateien steht in **HTML-Elementen**. HTML-Elemente werden durch so genannte **Tags** markiert. Fast alle HTML-Elemente werden durch ein einleitendes und ein abschließendes Tag markiert. Der Inhalt dazwischen ist der "Gültigkeitsbereich" des entsprechenden Elements. Tags werden in spitzen Klammern notiert.

Beispiel

```
<h1>HTML - die Sprache des Web</h1>
```

Erläuterung

Das Beispiel zeigt eine Überschrift 1. Ordnung. Das einleitende Tag `<h1>` signalisiert, dass eine Überschrift 1. Ordnung folgt (*h = heading = Überschrift*). Das abschließende Tag `</h1>` signalisiert das Ende der Überschrift. Ein abschließendes Tag beginnt mit einem Schrägstrich `"/"`.

Bei herkömmlichem HTML spielt es keine Rolle, ob die Elementnamen in den Tags in Klein- oder Großbuchstaben notiert werden. Dort bedeuten z.B. `<h1>` und `<H1>` das gleiche. In der neueren HTML-Variante, in XHTML, müssen die Elementnamen dagegen klein geschrieben werden. Das W3-Konsortium empfiehlt zwar für HTML aus Gründen der Lesbarkeit, Namen von Elementen groß zu schreiben. Im Hinblick auf eine spätere Deklarierung eines Dokuments als XHTML-Dokument ist diese Idee jedoch nicht unbedingt so gut und es ist besser, von vorneherein alle Elementnamen in Kleinbuchstaben zu schreiben.

Es gibt auch einige Elemente mit "Standalone-Tags", d.h. Elemente, die keinen Inhalt haben und deshalb nur aus einem Tag bestehen statt aus Anfangs- und End-Tag.

Beispiel

```
Eine Zeile, ein manueller Zeilenumbruch<br>
Und die nächste Zeile
```

Erläuterung

Am Ende der ersten Zeile signalisiert `<br>`, dass ein manueller Zeilenumbruch eingefügt werden soll (*br = break = Umbruch*).

Wenn man XHTML-gerecht schreiben wollte, müsste man Elemente mit Standalone-Tags anders notieren: anstelle von `<br>` müsste man dann `<br />` notieren - also den Elementnamen mit einem abschließenden Schrägstrich. Alternativ dazu könnte man auch `<br></br>` notieren, also ein Element mit Anfangs- und End-Tag, aber ohne Inhalt.

Verschachtelung von Elementen

Elemente können ineinander verschachtelt werden. Auf diese Weise entsteht eine hierarchische Struktur. Komplexere HTML-Dateien enthalten sehr viele Verschachtelungen. Deshalb sprechen Fachleute auch von **strukturiertem Markup**.

Beispiel

```
<h1><i>HTML</i> - die Sprache des Web</h1>
```

Erläuterung

Das `i`-Element steht für *italic* (= *kursiver Text*). Der Text zwischen `<i>` und `</i>` wird als kursiv interpretiert, abhängig von der eingestellten Schriftart und Schriftgröße für die Überschrift 1. Ordnung.

Versuche auch folgende Elemente:

`<b>...</b>`	zeichnet einen Text als fett aus
`<i>...</i>`	zeichnet einen Text als kursiv aus
`<tt>...</tt>`	zeichnet einen Text als dicktengleich formatiert aus (*tt* = *Teletyper* = *Fernschreiber*)
`<u>...</u>`	zeichnet einen Text als unterstrichen aus
`<strike>...</strike>`	zeichnet einen Text als durchgestrichen aus
`<s>...</s>`	zeichnet einen Text als durchgestrichen aus
`<big>...</big>`	zeichnet einen Text größer als normal aus
`<small>...</small>`	zeichnet einen Text kleiner als normal aus
`<sup>...</sup>`	zeichnet einen Text als hochgestellt aus
`<sub>...</sub>`	zeichnet einen Text als tiefgestellt aus

Attribute in Tags

Einleitende Tags und Standalone-Tags können zusätzliche Angaben enthalten.

Beispiel

```
<h1 align="center">HTML – die Sprache des Web</h1>
```

Erläuterung

Durch `align="center"` wird bewirkt, dass der Text zentriert ausgerichtet wird (*align* = *Ausrichtung*, *center* = *zentriert*).

Es gibt folgende Arten von Attributen in HTML-Elementen:

- Attribute mit Wertzuweisung, wobei es bestimmte erlaubte Werte gibt, z.B. bei `<h1 align="center">` (Überschrift 1. Ordnung zentriert ausgerichtet - hier sind nur die Werte `left`, `center`, `right` und `justify` erlaubt).
- Attribute mit freier Wertzuweisung, wobei jedoch ein bestimmter Datentyp oder eine bestimmte Konvention erwartet wird, z.B. `<style type="text/css">` (Bereich für Stylesheets definieren - hier wird ein so genannter Mime-Type als Wert erwartet, und Mime-Typen haben immer den Aufbau `Typ/Untertyp`). Oder `<table border="1">` (Tabelle mit Rahmen von 1 Pixel Stärke - hier wird eine numerische Angabe erwartet)

- Attribute mit freier Wertzuweisung ohne weitere Konventionen, z.B. `<p title="Aussage mit Vorbehalt">` - hier kann ein ganzer Text zugewiesen werden.
- Alleinstehende Attribute, z.B. `<hr noshade>` (Trennlinie ohne Schatten). Alleinstehende Attribute gibt es allerdings nur in herkömmlichem HTML. Wenn man XHTML-gerecht schreiben will, muss man `<hr noshade="noshade">` notieren.

Alle Werte, die du Attributen zuweist, müssen in Anführungszeichen stehen. Die meisten Browser nehmen es zwar nicht übel, wenn die Anführungszeichen fehlen, und das W3-Konsortium traf in der Vergangenheit auch schon recht unterschiedliche Aussagen darüber, doch seit dem HTML-Standard 4.0 sind die Anführungszeichen klipp und klar vorgeschrieben, und wer ordentliches HTML schreiben will, sollte sich daran halten.

Wie bei Elementnamen, so gilt auch bei Attributnamen: bei herkömmlichem HTML spielt es keine Rolle, ob die Attributnamen in Klein- oder Großbuchstaben notiert werden. In der neueren HTML-Variante, in XHTML, müssen die Attributnamen dagegen klein geschrieben werden. Bei den Wertzuweisungen an Attribute kann Groß- und Kleinschreibung abhängig von der Art des Wertes unterschieden werden oder auch nicht.

Neben Attributen, die nur in bestimmten HTML-Elementen vorkommen können, gibt es auch so genannte **Universalattribute**, die in vielen bzw. fast allen HTML-Elementen erlaubt sind.

Beispiel

```
<p id="Einleitung">Text</p>
```

Erläuterung

Das Beispiel definiert einen Textabsatz mit den HTML-Tags `<p>` und `</p>`. Im einleitenden `<p>`-Tag wird ein Universalattribut notiert, nämlich das Attribut `id=`. Damit kann man dokumentweit eindeutige Namen für einzelne HTML-Elemente vergeben.

HTML-Parser

Unter einem HTML-Parser versteht man eine Software, die HTML-Auszeichnungen erkennt in strukturierten Text umsetzt. Jeder Web-Browser verfügt über einen HTML-Parser, um überhaupt mit HTML klarzukommen. Solche HTML-Parser werden nun leider auf den meisten Webseiten mit Syntax-Fehlern in der Textauszeichnung konfrontiert. Oft sind es kleinere, nicht allzutragische Fehler, doch es gibt auch viele Web-Seiten, deren HTML-Quelltext nur das Prädikat "ungenügend" verdient, weil darin übelste Verunstaltungen der HTML-Regeln vorkommen. Strenge Parser, die genau gegen die HTML-Regeln prüfen, müssten die Umsetzung solcher Web-Seiten eigentlich abbrechen, und anstelle der Seite würden die Browser dann nur eine lapidare Fehlermeldung anzeigen. Da ein solcher Browser am breiten Markt jedoch keine Chance hätte, weil er kaum eine bekannte Web-Seite anzeigen würde, sind die HTML-Parser der heute verbreiteten Browser ziemlich gutmütige Wesen, die so ziemlich alles fressen, was ihnen vorgesetzt wird, und irgendetwas daraus machen, meistens sogar durchaus das, was der Autor der Web-Seite erreichen wollte. Am weitesten in dieser Kunst hat es der HTML-Parser des MS Internet Explorers gebracht. Das hat dem Internet Explorer einerseits den

Ruf beschert, "am besten" HTML zu können, aber Fachleute rümpfen aus dem gleichen Grund gerne die Nase über diesen Browser mit dem Argument, dass er durch sein Verhalten das schlampige und fehlerhafte Kodieren von HTML nur fördere.

Angesichts der wachsenden Komplexität der verschiedenen Sprachen, also HTML in Verbindung mit eingebettetem CSS, JavaScript, PHP usw., wird es immer wichtiger, die Syntax-Regeln von HTML einzuhalten.

Präsentation der Inhalte

Titel

Jede HTML-Datei sollte einen Titel erhalten. Das ist aus folgenden Gründen besonders wichtig:

- Der Titel der Datei wird bei der Anzeige im Web-Browser in der Titelzeile des Anzeigefensters angezeigt.
- Der Titel der Datei wird vom Web-Browser beim Setzen von Lesezeichen (Bookmarks, Favoriten) auf die Datei verwendet.
- Der Titel der Datei wird im Web-Browser in der Liste der bereits besuchten Seiten angezeigt.
- Der Titel der Datei dient im Web vielen automatischen Suchprogrammen als wichtiger Input. Wenn die Datei zu den Suchtreffern einer Suche gehört, bieten viele Suchmaschinen den Titel der Datei als anklickbaren Verweis an.

Beispiel

```
<head>
<title>Ausblick vom Hamburger Michel</title>
  <!-- ... andere Angaben im Dateikopf ... -->
</head>
```

Erläuterung

Innerhalb des Grundgerüsts einer HTML-Datei wird der Titel im Dateikopf notiert. `<title>` leitet die Angabe des Titels ein. Dahinter folgt der Text des Titels. Mit `</title>` wird die Titelangabe abgeschlossen (*title = Titel*).

Der Titeltext sollte nicht zu lang sein. Für den Titeltext gelten alle Regeln für Sonderzeichen und HTML-eigene Zeichen.

Überschriften

HTML unterscheidet 6 Überschriftenebenen, um Hierarchieverhältnisse in Dokumenten abzubilden.

Beispiel

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

```
<title>Text des Titels</title>
</head>
<body>
<h1>&Uuml;berschrift 1. Ordnung</h1>
<h2>&Uuml;berschrift 2. Ordnung</h2>
<h3>&Uuml;berschrift 3. Ordnung</h3>
<h4>&Uuml;berschrift 4. Ordnung</h4>
<h5>&Uuml;berschrift 5. Ordnung</h5>
<h6>&Uuml;berschrift 6. Ordnung</h6>
</body>
</html>
```

Erläuterung

`<h[1-6]>` (*h = heading = Überschrift*) leitet eine Überschrift ein. Die Nummer steht für die Überschriftenebene. 1 ist die höchste Ebene, 6 die niedrigste. Dahinter folgt der Text der Überschrift.
`</h[1-6]>` beendet die Überschrift und steht am Ende des Überschriftentextes.

Die Nummern bei einleitendem und abschließendem Tag müssen gleich sein.

Überschriften ausrichten

Überschriften werden linksbündig ausgerichtet, wenn du nichts anderes angibst. Du kannst eine Überschrift zentriert oder rechtsbündig ausrichten. Auch Blocksatz ist möglich.

Beispiel

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
       "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>
<h1 align="center">&Uuml;berschrift 1. Ordnung zentriert</h1>
<h2 align="center">&Uuml;berschrift 2. Ordnung zentriert</h2>
<h3 align="center">&Uuml;berschrift 3. Ordnung zentriert</h3>
<h1 align="right">&Uuml;berschrift 1. Ordnung rechtsb&uuml;ndig</h1>
<h2 align="right">&Uuml;berschrift 2. Ordnung rechtsb&uuml;ndig</h2>
<h3 align="right">&Uuml;berschrift 3. Ordnung rechtsb&uuml;ndig</h3>
</body>
</html>
```

Erläuterung

Durch die Angabe `align="center"` im einleitenden Überschriften-Tag erreichst du, dass die Überschrift zentriert ausgerichtet wird (*align = Ausrichtung, center = zentriert*). Mit der Angabe `align="right"` wird die Überschrift rechtsbündig ausgerichtet (*right = rechts*). Mit `align="justify"` erzwingst du den Blocksatz für die Überschrift (*justify = justieren*). Mit der Angabe `align="left"` kannst du die Normaleinstellung (linksbündige Ausrichtung) angeben.

Nicht alle Browser beherrschen den Blocksatz. Blocksatz ist im Zusammenhang mit Überschriften auch nur bedingt praxistauglich, da Blocksatz erst bei mehrzeiligen Texten

zum Tragen kommt.

align ist im HTML-4-Standard als *deprecated* (missbilligt) eingestuft. Stattdessen wird empfohlen, CSS Stylesheets zu benutzen,
z.B.: `<h1 style="text-align:center">...</h1>`.

Textabsätze

Absätze dienen der optischen Gliederung eines Textes. Beim Erstellen von HTML-Dateien genügt es nicht, im Editor einen harten Umbruch einzufügen. Wie du bereits weißt, ignorieren WWW-Browser solche Umbrüche (Stichwort: *white space*).

Beispiel

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
       "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>
<h1>Textabs&auml;tze definieren</h1>
<p>Hier beginnt ein Absatz, und hier ist er zu Ende.</p>
<p>Hier beginnt ein neuer Absatz, und hier ist er zu Ende.</p>
</body>
</html>
```

Erläuterung

`<p>` (*p = paragraph = Absatz*) leitet einen Textabsatz ein. `</p>` beendet den Textabsatz und steht am Ende des Absatztextes.

Alleinstehende `<p>`-Tags, wie sie früher mal zulässig waren, sind mittlerweile nicht mehr HTML-gerecht. Notiere Textabsätze immer mit einleitendem und abschließendem Tag.

Das `<p>`-Element darf keine anderen blockerzeugenden Elemente wie z.B. Überschriften, Textabsätze, Listen, Zitate oder Adressen enthalten.

Textabsätze ausrichten

Textabsätze werden linksbündig ausgerichtet, wenn du nichts anderes angibst. Du kannst einen Textabsatz auch zentriert oder rechtsbündig ausrichten. Auch Blocksatz ist möglich.

Beispiel

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
       "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>
<h1>Textabs&auml;tze ausrichten</h1>
<p align="center">Dies ist ein zentrierter Absatz.</p>
<p align="right">Dies ist ein Absatz, der rechts ausgerichtet ist.</p>
```

```
</body>
</html>
```

Erläuterung

Durch die Angabe `align="center"` im einleitenden <p>-Tag erreichst du, dass der Textabsatz zentriert ausgerichtet wird (*align = Ausrichtung, center = zentriert*). Mit `align="right"` wird der Absatz rechtsbündig ausgerichtet (*right = rechts*). Mit `align="justify"` erzwingst du den Blocksatz für den Absatz (*justify = justieren*). Mit `align="left"` kannst du die Normaleinstellung (linksbündige Ausrichtung) angeben.

Nicht alle Browser beherrschen den Blocksatz.

`align` ist im HTML-4-Standard als *deprecated* (missbilligt) eingestuft. Stattdessen wird empfohlen, CSS Stylesheets zu benutzen,
z.B.: `<p style="text-align:center">...</p>`.

Zeilenumbrüche

Text innerhalb von normalen Absätzen, Listen, sowie in Überschriften oder Tabellenzellen wird vom Web-Browser bei der Anzeige automatisch umgebrochen. Du kannst jedoch an einer gewünschten Stelle einen Zeilenumbruch erzwingen.

Beispiel

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
       "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>
<h1>Otto's Brief an die Mutter</h1>
<p>Liebe Mama!</p>
<p>Vor drei Wochen habe ich erfahren<br>
dass Du krank bist<br>
Mach Dir keine Sorgen<br>
Mir geht es gut.</p>
</body>
</html>
```

Erläuterung

`<br>` (*br = break = Umbruch*) fügt an der gewünschten Stelle einen Zeilenumbruch ein. Dabei ist es egal, ob das allein stehende Tag am Ende der vorherigen Zeile steht (wie im Beispiel), oder in einer eigenen Zeile, oder am Anfang der folgenden Zeile.

Wenn du **XHTML-konform** arbeiten willst, musst du das `br`-Element als inhaltsleer kennzeichnen. Dazu notiere das allein stehende Tag in der Form `<br />`.

Automatischen Zeilenumbruch verhindern

Du kannst einen Textbereich bestimmen, in dem kein automatischer Zeilenumbruch erfolgt. Alles, was innerhalb dieses Bereichs steht, wird in einer langen Zeile angezeigt.

Der Anwender kann dann mit der horizontalen Scroll-Leiste die überlange Textzeile anzeigen.

Diese Möglichkeit gehört jedoch nicht zum offiziellen HTML-Sprachstandard. Du solltest sie daher vermeiden.

Beispiel

```
<html>
<head>
<title>Text des Titels</title>
</head>
<body>
<h1>ARD und ZDF</h1>
<nobr>Die vom ZDF sagen die vom ARD senden st&auml;ndig Wiederholungen,
und die vom ARD sagen die vom ZDF senden st&auml;ndig Wiederholungen,
und so wiederholen sich ARD und ZDF st&auml;ndig ohne überhaupt etwas zu
senden.</nobr>
</body>
</html>
```

Erläuterung

`<nobr>` bewirkt, dass der auf das Tag folgende Text nicht umgebrochen wird (*nobr = no break = kein Umbruch*). Am Ende des Textabschnitts, der nicht umgebrochen werden soll, notiere das abschließende Tag `</nobr>`.

Wenn du Textzeilen unabhängig vom Anzeigefenster des Anwenders genau kontrollieren und nach HTML-Standard arbeiten willst, kannst du präformatierten Text einsetzen. (`<pre>` und `</pre>`)

Geschützte Leerzeichen

Du kannst verhindern, dass bei einem Leerzeichen ein automatischer Zeilenumbruch erfolgen darf.

Beispiel

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
       "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>
<h1>HTML 2.0 und HTML 4.0</h1>
<p>Es gibt vieles, worin sich HTML 2.0 und HTML 4.0
unterscheiden.</p>
</body>
</html>
```

Erläuterung

Die Zeichenfolge ` ` erzeugt ein geschütztes Leerzeichen (*nbsp =nonbreaking space = nicht umbrechbares Leerzeichen*). Es wird ein normales Leerzeichen angezeigt, doch an dieser Stelle kann kein Zeilenumbruch erfolgen.

Die gleiche Wirkung erzielst du durch Notieren der Zeichenfolge ` `.

Zeilenumbruch erlauben

Web-Browser brechen Text normalerweise nur bei Leerzeichen um, weil durch Leerzeichen Wörter voneinander abgegrenzt werden. Du kannst explizit weitere Stellen markieren, an denen er den Text umbrechen darf. Dies gilt für alle Absatzarten in HTML.

Diese Möglichkeit gehört jedoch nicht zum offiziellen HTML-Sprachstandard. Du solltest sie daher vermeiden.

Beispiel

```
<html>
<head>
<title>Text des Titels</title>
</head>
<body>
<h1>Langes Wort</h1>
<p>Donaudampfschiffahrts-<wbr>Kapitänsmütze Donaudampfschiffahrts-
<wbr>Kapitänsmütze ... </p>
</body>
</html>
```

Erläuterung

Mit `<wbr>` markierst du eine Stelle, an der getrennt werden darf, falls diese Stelle bei der Bildschirmanzeige am Ende der Zeile steht (*wbr = word break = Umbruch innerhalb eines Wortes*). Sinnvoll ist dies bei langen Wörtern oder aus Bindestrichen bestehenden Ausdrücken.

Innerhalb von Abschnitten mit verhindertem Zeilenumbruch bewirkt `<wbr>`, dass an der betreffenden Stelle trotzdem ein Umbruch erfolgen darf.

Farben

Farben kannst du in HTML in vielen Zusammenhängen definieren, zum Beispiel bei:

- dateiweiten Hinter- und Vordergrundfarben
- bei Schriftfarben für Textabschnitte
- bei Hintergrundfarben in Tabellen

Allerdings sind all diese Angaben in HTML vom W3-Konsortium mittlerweile als *deprecated* (missbilligt) gekennzeichnet, d.h. sie sollen künftig nicht mehr zum Sprachstandard gehören. Der Grund ist, dass sich all diese Farben auch mit Hilfe von CSS Stylesheets definieren lassen. Denn CSS ist die Sprache für die Optik, nicht mehr HTML. Allerdings lohnt es sich trotzdem, mit dem Definieren von Farben in HTML zu befassen, weil in CSS die gleichen Farbangaben möglich sind, dort aber darüber hinaus noch weitere Möglichkeiten bestehen.

Grundsätzlich gibt es zwei Möglichkeiten, Farben in HTML zu definieren:

- durch Angabe der RGB-Werte der gewünschten Farbe in Hexadezimalform (RGB = Rot/Grün/Blau-Wert der Farbe)
- durch Angabe eines Farbnamens

Wenn du hexadezimale Werte angibst, arbeitest du Browser-unabhängig, und du hast die volle Freiheit zwischen 16,7 Millionen Farben.

Wenn du Farbnamen angibst, umgehst du die etwas schwierige Definition einer Farbe im Hexadezimal-Modus. Derzeit sind jedoch nur 16 Farbnamen offiziell standardisiert. Weitere Farbnamen gibt es, sie sind jedoch Browser-abhängig.

Hexadezimale Angabe von Farben

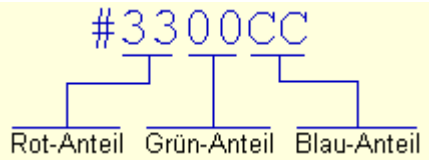
Wenn du Farben direkt im Hexadezimal-Modus definierst, musst du die gewünschte Farbe aus Angaben zu den drei Grundfarben **R**ot, **G**rün und **B**lau (RGB-Werte) zusammenstellen.

Beispiele

```
<body bgcolor="#808080"> <!-- dunkelgrauer Dateihintergrund -->
<table bgcolor="#00C0C0"> <!-- blaugrüner Tabellenhintergrund -->
<hr color="#CC00CC">  <!-- violette Trennlinie -->
```

Erläuterung

Jede hexadezimale Farbdefinition ist 6stellig und hat das Schema: #XXXXXX.
Zunächst notiere also ein Gatter #. Dahinter folgen 6 Stellen für die Farbdefinition. Die ersten beiden Stellen stellen den Rot-Wert der Farbe dar, die zweiten beiden Stellen den Grün-Wert, und die letzten beiden Stellen den Blau-Wert.



Hexadezimale Ziffern sind:

Hexadezimal Ziffer	Dezimalwert
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
A	10
B	11
C	12

D	13
E	14
F	15

Eine hexadezimale Ziffer kann also 16 Zustände haben. Für jeden Farbwert (Rot, Grün, Blau) stehen 2 Ziffern zur Verfügung. Das macht 16 x 16 (= 256) mögliche Zustände pro Farbwert.

Es gibt 16 Grundfarben, die von jedem VGA-kompatiblen Bildschirm angezeigt werden können.

Farbnamen für die 16 Grundfarben

Um eine Farbe mit Hilfe eines Farbnamens zu definieren, gib anstelle des hexadezimalen RGB-Werts einfach den gewünschten Farbnamen an. Die folgenden Farbnamen sind Bestandteil von HTML seit Version 3.2 und werden von vielen WWW-Browsern verstanden.

Beispiele

```
<body bgcolor="black"> <!-- schwarzer Dateihintergrund -->
<table bgcolor="aqua"> <!-- hellblauer Tabellenhintergrund -->
<hr color="red">  <!-- rote Trennlinie -->
```

Erläuterung

Gib den gewünschten Farbnamen an einer Stelle an, an der eine Farbangabe erlaubt ist.

Hier ist eine Übersicht über die definierten Farbnamen:

Black	#000000	gray	#808080
Maroon	#800000	red	#FF0000
Green	#008000	lime	#00FF00
Olive	#808000	yellow	#FFFF00
Navy	#000080	blue	#0000FF
Purple	#800080	fuchsia	#FF00FF
Teal	#008080	aqua	#00FFFF
Silver	#C0C0C0	white	#FFFFFF

Farbe für den Hintergrund

Du kannst eine Farbe für den Hintergrund des Anzeigefensters bestimmen. Die gesamte HTML-Datei wird auf dieser Hintergrundfarbe angezeigt.

Beispiel

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
       "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body bgcolor="#CCFFFF">
<h1>Text, Verweise, Grafikreferenzen usw.</h1>
</body>
</html>
```

Erläuterung

Die Angabe zur Hintergrundfarbe erfolgt im einleitenden `<body>`-Tag der HTML-Datei. Mit dem Attribut `bgcolor=` bestimmst du die Farbe für den Bildschirmhintergrund (*bgcolor = background color = Hintergrundfarbe*). Die gesamte HTML-Datei wird auf der hier definierten Hintergrundfabe angezeigt.

Das Attribut bgcolor= ist als *deprecated* (missbilligt) eingestuft und soll künftig vermieden werden. Den gleichen Effekt erreichst du nämlich auch mit Hilfe von CSS Stylesheets, z.B. so:
`<body style="background-color:#CCFFFF">`

Wenn du ein Dokument aus mehreren HTML-Dateien erstellst, das einen einheitlichen Hintergrund und Textvordergrund haben soll, musst du die Hintergrundfarbe in jeder HTML-Datei neu definieren. Natürlich kannst du auch für jede HTML-Datei andere Farben definieren.

Wenn du eine Hintergrundfarbe definierst, solltest du auch passende (kontrastierende) Textvordergrundfarben definieren (siehe hierzu folgender Abschnitt).

Farben für Text und Verweise

Du kannst dateiweite Farben definieren:

- für Text (gültig für alle Elemente wie Überschriften, normalen Fließtext, Listen usw.),
- für Verweise zu noch nicht besuchten Stellen,
- für Verweise zu bereits besuchten Stellen,
- für Verweise, während sie angeklickt werden.

Beispiel

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

```
<body bgcolor="#663333" text="#FFCC99" link="#FF9966" vlink="#FF9900"
alink="#FFFFFF">
<h1>Text</h1>
<a href="http://www.yahoo.de/">Verweis zu Yahoo</a>
</body>
</html>
```

Erläuterung

Die Angaben erfolgen im einleitenden `<body>`-Tag der HTML-Datei.

Mit `text=` definierst du eine Farbe für den Text.
Mit `link=` definierst du eine Farbe für Verweise zu noch nicht besuchten Dateien (*link = Verweis*).
Mit `vlink=` definierst du eine Farbe für Verweise zu bereits besuchten Dateien (*vlink = visited link = besuchter Verweis*).
Mit `alink=` definierst du eine Farbe für Verweise, die der Anwender gerade anklickt (*alink = activated link = aktivierter Verweis*).

Alle diese Attribute sind als *deprecated* (missbilligt) eingestuft und sollen künftig vermieden werden. Den gleichen Effekt erreichst du nämlich auch mit Hilfe von CSS Stylesheets, z.B. im Dateikopf zwischen `<head>` und `</head>` mit dem folgenden Quelltext:

```
<style type="text/css">
body { background-color:#663333; color:#FFCC99; }
a:link { color:#FF9966; }
a:visited { color:#FF9900; }
a:active { color:#FFFFFF; }
</style>
```

Hintergrundbild

Du kannst für die Anzeige einer HTML-Datei ein Hintergrundbild bestimmen. Dabei wird das Bild über das ganze Anzeigefenster hinweg immer wiederholt, so dass ein Tapeteneffekt (Wallpaper) entsteht. Besonders geeignet für Wallpaper-Effekte sind relativ kleine Grafiken, die irgendein abstraktes Muster darstellen.

Die Hintergrundgrafik sollte als Grafikdatei im GIF-Format oder JPEG-Format vorliegen.

Beispiel

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
       "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body background="background.jpg" text="#990000" link="#0000CC"
vlink="#000066" alink="#000000">
<h1>Text</h1>
<a href="http://www.yahoo.de/"><b>Verweis zu Yahoo</b></a>
</body>
</html>
```

Erläuterung

Die Angabe zum Einbinden eines Hintergrundbildes erfolgt im einleitenden `<body>`-Tag der HTML-Datei. Mit dem Attribut `background=` bestimmst du eine Grafikdatei als Hintergrundbild (*background = Hintergrund*). Die gesamte HTML-Datei wird auf dem hier definierten Hintergrundbild angezeigt.

Das Attribut `background=` ist als *deprecated* (missbilligt) eingestuft und soll künftig vermieden werden. Den gleichen Effekt erreichst du nämlich auch mit Hilfe von CSS Stylesheets, z.B. so:

```
<body style="background-image:url(background.jpg)">
```

Das Hintergrundbild gilt jeweils für die HTML-Datei, in der es definiert wird. Wenn du ein Dokument aus mehreren HTML-Dateien erstellst, das einen einheitlichen Hintergrund haben soll, musst du die Angabe in jeder HTML-Datei wiederholen.

Wenn du ein Hintergrundbild definierst, solltest du auch passende Farben für Text und Verweise definieren.

Verwende bei textorientierten WWW-Seiten unauffällige Hintergrundbilder, bei denen die Lesbarkeit des Textes nicht leidet. Auffällige Hintergrundfarben solltest du nur verwenden, wenn die gesamte WWW-Seite grafisch aufgebaut ist und wenn die Grafiken im Vordergrund zu dem auffälligen Hintergrund passen.

Hintergrundmusik einbinden

Du kannst bestimmen, dass beim Aufruf einer HTML-Datei eine Hintergrundmusik ertönt. Dazu gibt es zwei Lösungen. Beide der im Folgenden beschriebenen Lösungen - die eine für Microsoft's Internet Explorer, die andere für Netscape - sind proprietär. Dazu kommt, dass Netscape 4.x in einigen Zwischenversionen dabei versagt. Beide Lösungen gehören nicht zum HTML-Standard.

Ferner solltest du dir beim Wunsch, Hintergrundmusik einzubinden, darüber im Klaren sein, dass die Mehrzahl der Anwender im Web genervt ist von dem Gedudel und sehr schnell wieder von solchen Seiten verschwindet.

Beispiel

```
<html>
<head>
<title>Text des Titels</title>
<!-- Microsoft: -->
<bgsound src="background.mid" loop="infinite">
</head>
<body>
<!-- Netscape: -->
<embed src="background.mid" autostart="true" loop="true" hidden="true"
height="0" width="0">
<h1>Inhalt der Seite</h1>
</body>
</html>
```

Erläuterung

Mit `<bgsound ...>` definierst du im Kopf einer HTML-Datei eine Hintergrundmusik nach Microsoft-Syntax für die Anzeige der Datei (*bgsound = background sound =*

Hintergrundmusik). Mit `<embed ...>` erreichst du das Gleiche für Netscape (*embed = einbetten*). Hinter der Angabe `src=` folgt in beiden Fällen die Angabe der gewünschten Musikdatei (*src = source = Quelle*). Es sollte sich möglichst um Dateien der Typen MID, AU oder WAV handeln. Um eine sichtbare Anzeige des Abspiel-Players zu unterdrücken, sind bei der Netscape-Syntax die Angaben `hidden="true" height="0" width="0"` erforderlich (*width = Breite, height = Höhe, hidden = versteckt*). Auch den automatischen Start des Abspielvorgangs musst du in der Netscape-Syntax explizit angeben - durch `autostart="true"`. Schließlich kannst du noch bestimmen, ob die Musikdatei nur einmal, mehrmals oder endlos oft (bis zum Aufrufen einer anderen HTML-Datei) abgespielt wird. Nach der Microsoft-Syntax kannst du mit `loop="infinite"` eine Endloswiederholung erzwingen. Wenn du die Wiederholungszahl begrenzen willst, gib anstelle von `infinite` einfach die gewünschte Anzahl der Wiederholungen an. Bei der Netscape-Syntax hast du nur die Wahl zwischen Endloswiederholung und keiner Wiederholung. Um eine Endloswiederholung zu erzwingen, notiere die Angabe `loop="true"`. Ansonsten lasse die Angabe einfach weg.

Im obigen Beispiel wird vorausgesetzt, dass sich die Musikdatei im gleichen Verzeichnis befindet wie die HTML-Datei. Wenn die Datei in einem anderen Verzeichnis steht, musst du den relativen oder absoluten Pfadnamen angeben.

Die Ausgabe einer Hintergrundmusik setzt beim Anwender natürlich entsprechende Hardware (Soundkarte, Lautsprecher) voraus. Ferner muss der Web-Browser die Möglichkeit haben, die Ausgabe der Musikdatei zu steuern.

Allgemeines zum Referenzieren in HTML

HTML-Dateien bestehen bekanntlich nur aus Text. Dennoch enthalten viele Web-Seiten Grafiken, Hintergrundgrafiken, Multimedia-Elemente, Java-Applets, Flash-Animationen und dergleichen. Solche Elemente werden in HTML in Form einer Referenz auf eine entsprechende Datenquelle notiert. Auch ein ausführbarer Verweis zu einer anderen eigenen oder fremden Web-Seite ist nur ausführbar, wenn er sein Verweisziel benennt. Für all diese Zwecke wird das Referenzieren in HTML benötigt.

Ebenso gibt es in Ergänzungssprachen wie CSS Stylesheets oder JavaScript Stellen, an denen du andere Datenquellen referenzieren musst.

Die Regeln zum Referenzieren sind dabei immer die gleichen. Der Grund ist das zentrale und einheitliche Adressierungs-Schema im Web, das unabhängig von der Syntax einzelner Betriebssysteme gilt und die genaue Adressierung beliebiger Quellen im Web erlaubt.

Mit vollständigen URIs referenzieren

Mit vollständigen URIs musst du dann referenzieren, wenn sich die gewünschte Datenquelle grob gesagt nicht im aktuellen eigenen Web-Angebot befindet.

Ein **URI** (*Universal Resource Identifier - universelle Quellenbezeichnung*) ist beispielsweise so etwas wie http://www.teamone.de/ oder http://selfhtml.teamone.de/html/allgemein/referenzieren.htm. Beide Beispieladressen sind aber gleichzeitig auch so genannte **URL**s (*Uniform Resource Locators - einheitliche Quellenorter*). Und dann gibt es - um die Verwirrung komplett zu machen - auch noch so

genannte **URN**s (*Uniform Resource Names - einheitlichen Quellennamen*). Letztere sind dazu gedacht, um nicht wirklich existierende Datenquellen oder Quellen, die zwar existieren, aber durch kein bekanntes Internet-Protokoll im Netz abrufbar sind, dennoch eindeutig zu benennen. Ein URI ist also der Oberbegriff für URL und URN, wobei URI und URL bei typischen Adressen, hinter denen sich konkrete Dateien oder Datenquellen verbergen, und um die es hier geht, faktisch das Gleiche sind. Im HTML-Standard wird aber von URIs geredet.

Beispiele für URIs

```
http://www.ihr-guter-name.de/
http://www.ihr-guter-name.de/index.htm
http://www.ihr-guter-name.de/index.htm#impressum
http://www.ihr-guter-name.de/hintergrund.gif
http://www.ihr-guter-name.de/praesentation.pdf
http://www.ihr-guter-name.de/cgi-bin/suche.cgi?ausdruck=Hasenjagd
http://www.google.com/search?hl=de&safe=off&q=Stefan+M%FCnz&lr=
ftp://www.ihr-guter-name.de/praesentation.pdf
http://192.168.78.10/
http://www.ihr-guter-name.de:8082/geheim.htm
```

Erläuterung

Ein vollständiger URI besteht aus der Angabe eines Internet-Protokolls, z.B. `http` oder `ftp`, gefolgt von einem Doppelpunkt. Dahinter kann - das ist von Protokoll zu Protokoll verschieden - eine Zusatzangabe zu einem lokalen Netzwerknamen möglich sein. Diese Angabe wird in zwei Schrägstriche `//` eingeschlossen. Bei den meisten Adressen gibt es keine solche Angabe, weshalb die beiden Schrägstriche dort einfach ohne Inhalt nebeneinander stehen.

Hinter diesen Angaben folgt die Adresse des Hostrechners im Netz, auf dem sich die Datenquelle befindet. Das kann ein Domain-Name oder eine numerische IP-Adresse sein.

Hinter der Adressierung des Hostrechners kann - durch einen Doppelpunkt abgetrennt, eine so genannte Portnummer folgen, wie im letzten der obigen Beispiele bei `:8082`. Das ist immer dann erforderlich, wenn die Datenquelle nicht über den Standard-Port des angegebenen Protokolls wie etwa `http` erreichbar ist, sondern über einen anderen Port. In der Praxis benötigst du die Portangabe eher selten, aber kennen sollten du sie.

Dahinter folgt schließlich die lokale Pfadangabe zur gewünschten Datenquelle. Egal um welches Betriebssystem es sich dabei handelt - Verzeichnispfade werden stets durch einfache Schrägstriche getrennt. Es ist Aufgabe der Server-Software auf dem Rechner, die Pfadangaben korrekt aufzulösen. Auf diese Weise brauchst du dir keine Gedanken zu machen, welches System der angesprochene Rechner benutzt.

Auf dem Rechner können beliebige Dateien und Datenquellen angesprochen werden. Voraussetzung ist, dass sie über das angegebene Protokoll wie z.B. `http` unter der Adressierung erreichbar sind. Es muss sich nicht unbedingt um Dateien handeln. So kann mit # und einem Namen dahinter etwa ein bestimmter Zielanker in einer HTML-Datei angesprochen werden. Wie solche Zielanker definiert werden, ist im Abschnitt Anker definieren und Verweise zu Ankern beschrieben. Auch Aufrufe von CGI-Scripts mit Parametern sind URIs, wie im obigen Beispiel `suche.cgi?ausdruck=Hasenjagd`.

Zeichen, die nicht zum ASCII-Zeichensatz gehören oder in URIs Bedeutung haben (z.B. der Schrägstrich, der Doppelpunkt oder das Prozentzeichen) musst du innerhalb von URIs maskieren. Das geschieht durch Angabe eines Prozentzeichens `%` mit anschließendem Hexadezimalwert für das Zeichen. Im obigen Beispiel siehst du das z.B. bei `M%FCnz`, wobei `FC` die hexadezimale Angabe der Zahl 252 ist und diese wiederum den Buchstaben "ü" bedeutet.

Datenquellen im eigenen Web-Angebot kannst du natürlich auch mit vollständigen URIs referenzieren. Damit schränkst du dich jedoch ein (siehe die einleitenden Bemerkungen zum Abschnitt).

Mit absoluten Pfadangaben relativ zum Basis-URI referenzieren

Diese Variante der Referenzierung kannst du wählen, wenn die gewünschte Datenquelle auf dem gleichen Hostrechner liegt und über das aktuelle Protokoll und den Standard-Port erreichbar ist. Das klingt komplizierter als es ist. In dem vollständigen URI http://selfhtml.teamone.de/html/allgemein/referenzieren.htm ist der Teil /html/allgemein/referenzieren.htm eine absolute Pfadangabe relativ zur Basis-URI http://selfhtml.teamone.de. Innerhalb des eigenen Web-Angebots und der eigenen Domain oder Sub-Domain kannst du also mit solchen Pfadangaben arbeiten.

Beispiele

```
/
/index.htm
/index.htm#impressum
/hintergrund.gif
/praesentation.pdf
/cgi-bin/suche.cgi?ausdruck=Hasenjagd
/search?hl=de&safe=off&q=Stefan+M%FCnz&lr=
```

Erläuterung

Der erste Schrägstrich hinter dem Basis-URI steht für das Wurzelverzeichnis des jeweiligen Internet-Services. Es handelt sich meistens nicht um das tatsächliche Wurzelverzeichnis des Rechners oder der Festplatte, auf die du da zugreifst. Bei Web-Servern ist beispielsweise einstellbar, welches tatsächliche Verzeichnis dem Web-Wurzelverzeichnis entsprechen soll.

Mit relativen Pfadangaben relativ zum Basis-URI referenzieren

Diese Variante kannst du wählen, wenn du den jeweils aktuellen URI als Bezugs-URI wählst. Dann kannst du von hieraus relativ adressieren. Die HTML-Datei mit dem URI http://selfhtml.teamone.de/html/allgemein/referenzieren.htm enthält beispielsweise zum Referenzieren einer Grafik folgende Angabe: ../../src/logo.gif. Das bedeutet: gehe zwei Verzeichnisse nach oben, von dort aus ins Unterverzeichnis src und dort findest du die Datei logo.gif. Absolut gesehen hat diese Datei also den URI http://selfhtml.teamone.de/src/logo.gif.

Diese Form der relativen Adressierung ist innerhalb von Web-Projekten sehr zu empfehlen. Der Grund ist, dass du das Web-Projekt auf diese Weise problemlos an eine andere Adresse verschieben kannst, und trotzdem funktionieren noch alle projektinternen

Verweise und Grafikreferenzen. Gerade wenn du dein Projekt auch mal auf CD-ROM oder anderen Medien veröffentlichen willst, ist die relative Adressierung ein „Muss“.

Beispiele

```
./
farben.htm
./farben.htm
bilder/grafik.gif
./bilder/grafik.gif
../
../../../../woanders/datei.htm
```

Erläuterung

Eine Datei im gleichen Verzeichnis wie dem aktuellen kannst du einfach durch Angabe des Dateinamens referenzieren - im obigen Beispiel etwa die Datei `farben.htm`. Das aktuelle Verzeichnis referenzierst du durch `./` - also einem Punkt, gefolgt von einem Schrägstrich. Die Adressierung von `farben.htm` und `./farben.htm` im obigen Beispiel hat also den gleichen Effekt.

Eine Angabe wie `bilder/grafik.gif` referenziert eine Datei namens `grafik.gif` im Verzeichnis `bilder`, das ein Unterverzeichnis des aktuellen Verzeichnisses ist. Die Notation `./bilder/grafik.gif` hat wieder den gleichen Effekt wie `bilder/grafik.gif`.

Mit `../` referenzierst du das Verzeichnis über dem aktuellen Verzeichnis, egal wie es heißt. Mit `../../` referenzierst du das Verzeichnis über dem Verzeichnis über dem aktuellen Verzeichnis usw. Von jedem der so adressierten Verzeichnisse kannst du wieder auf deren Unterverzeichnisse zugreifen, wie im letzten der obigen Beispiele gezeigt.

Verweise zu Dateien oder Quellen im Projekt

Ein Web-Projekt besteht typischerweise aus mehreren bis vielen Einzelseiten, die miteinander verlinkt sind.

Um Verweise auf andere Projektdateien zu definieren, empfiehlt es sich, relative Angaben zum Verweisziel zu machen. Das Projekt bleibt dadurch flexibler, und die Verweise funktionieren auch in anderen Umgebungen (z.B. solange du das Projekt lokal auf deinem PC erstellen und austesten willst, oder wenn du es mal auf CD-ROM präsentieren möchtest).

Beispiel Teil 1 - index.htm

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
       "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Homepage</h1>
<p>Dies ist eine ganz einfache Homepage
```

```
mit einem <a href="zweiteseite.htm">Verweis zu einer anderen Seite des
Projekts</a>.
Eine <a href="../../tabellen/anzeige/layout.htm">aufwendigere Homepage
mit projekt-internen Verweisen</a> k&ouml;nnen Sie aber ebenfalls
aufrufen.</p>

</body>
</html>
```

Beispiel Teil 2 - zweiteseite.htm

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
       "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<div style="width:100%; border:thin solid gray; padding:5px">
<a href="index.htm">Homepage</a>
</div>

<h1>Zweite Seite</h1>
<p>Ja, hurra, geschafft!</p>

</body>
</html>
```

Erläuterung

Das Beispiel zeigt zwei kleinere HTML-Dateien. Die erste - mit Namen index.htm - enthält einen Verweis zur zweiten Datei. Der Dateiname der zweiten Datei lautet zweiteseite.htm. Beide Dateien sind im gleichen Verzeichnis abgelegt. Deshalb genügt bei `href=` die Angabe des Dateinamens ohne weitere Zusätze. Im Beispiel der Datei index.htm wird aber auch noch gezeigt, wie Verweise zu Dateien in anderen Verzeichnissen definiert werden.

Die zweite Datei im Beispiel (zweiteseite.htm) enthält einen typischen "Rückverweis" auf die erste Seite, also auf die "Einstiegsseite". Ein Rückverweis ist kein spezieller Verweis, sondern einfach wieder ein Verweis auf die gewünschte Datei. Da beide Dateien im gleichen Verzeichnis liegen, genügt auch in diesem Fall wieder die Angabe des Dateinamens, also `href="index.htm".`

Der "Rückverweis" in der zweiten Datei wird im obigen Beispiel mit Hilfe eines allgemeinen Bereichs, dem ein paar CSS-Formate zugewiesen sind, optisch etwas aufgewertet. Er steht am Anfang des sichtbaren Dateikörpers und durch den Rahmen, den er dank CSS erhält, wirkt der Bereich wie eine Navigationsleiste. Es ist sehr empfehlenswert, solche typischen Navigationsverweise immer an der gleichen Stelle einer Seite zu notieren und optisch so zu gestalten, dass der Navigationszweck ersichtlich ist. HTML bietet keine speziellen Verweise oder gestalterischen Möglichkeiten für Navigationsleisten und Navigationsverweise an. Navigationsleisten musst du mit den zur Verfügung stehenden Bordmitteln selbst kreieren. (Genaugenommen ist diese Aussage falsch: HTML bietet sehr wohl so etwas an, nämlich mit der Möglichkeit, logische Beziehungen zu definieren. Leider wird dies jedoch von den Browsern bis heute nicht unterstützt.)

Adressbasis

Du kannst innerhalb einer HTML-Datei, die du auf einem WWW-Server ablegst, nochmals deren internet-weit eindeutigen, genauen URL notieren. Ein Web-Browser, der diese Information ausliest, kann in Fehlersituationen besser auf verknüpfte oder referenzierte Dateien zugreifen. Bei Projektverweisen zu anderen HTML-Dateien und bei Referenzen von Grafiken oder Multimedia gilt die hier definierbare Basis als Bezug.

Beispiel

```
<head>
<base href="http://selfaktuell.teamone.de">
  <!-- ... andere Angaben im Dateikopf ... -->
</head>
```

Erläuterung

Die Angabe der Dateibasis erfolgt im Dateikopf mit `<base href= ... >` und dem genauen URL der Datei (*base = Basis, href = hyper reference = Hyper(text)-Referenz*).

Wenn nun in der Datei beispielsweise folgende Grafikreferenz steht:
`<img src="/src/logo.gif">`
... so ermittelt der WWW-Browser diese Grafik mit dem absoluten URL:
`<img src="http://selfaktuell.teamone.de/src/logo.gif">`.

Wenn es wichtig für dich ist, dass deine Projekte leicht auf andere Server-Rechner, in andere Verzeichnisstrukturen usw. übertragbar bleiben, solltest du auf die Definition einer Adressbasis verzichten. Denn mit der Angabe einer absoluten Adressbasis wird der WWW-Browser stets versuchen, Verweisziele und referenzierte Dateien von der hier angegebenen Adresse zu laden. Dies macht das Austesten der Dateien auf einem lokalen PC unmöglich!

Die Angabe der Adressbasis ist dagegen ein gewisser Schutz vor HTML-unkundigen Anwendern, die sich eine HTML-Datei lokal abspeichern oder das Cache-Verzeichnis ihres WWW-Browsers durchsuchen. Beim lokalen Aufruf der HTML-Datei wird der WWW-Browser stets eine Online-Verbindung verlangen und die Datei von der angegebenen Adresse laden wollen.

Anker definieren und Verweise zu Ankern

Du kannst innerhalb einer HTML-Datei Anker definieren. Dann kannst du Verweise zu solchen Ankern setzen, um einen Sprung genau an die Ankerstelle innerhalb der Datei zu veranlassen. Der Verweis kann in der gleichen Datei stehen. Dann wird einfach ein Sprung innerhalb der angezeigten Seite ausgeführt. Der Verweis kann aber auch in einer anderen Datei stehen. Dann wird die Zieldatei geladen, und der Browser springt, sobald er die Stelle mit dem Anker geladen hat, an die entsprechende Stelle innerhalb der Datei.

Beispiel

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

```
<head>
<title>Text des Titels</title>
</head>
<body>

<h1><a name="anfang">Lange Seite</a></h1>

<p><a href="#kap02">Kapitel 2</a></p>

<hr><hr><hr><hr><hr><hr><hr><hr><hr><hr><hr><hr><hr><hr><hr><hr><hr>
<!-- usw. lauter Trennlinien, um Raum zu schaffen -->

<h2><a name="kap02">Kapitel 2</a></h2>

<p><a href="#anfang">Seitenanfang</a> oder
<a href="../projektintern.htm#anker">Anker definieren und Verweise zu
Ankern</a></p>

<hr><hr><hr><hr><hr><hr><hr><hr><hr><hr><hr><hr><hr><hr><hr><hr><hr>
<!-- usw. lauter Trennlinien, um Raum zu schaffen -->

</body>
</html>
```

Erläuterung

Ein Anker wird genau wie ein Verweis mit Hilfe des `a`-Elements erzeugt. Der Unterschied besteht darin, dass kein Attribut `href=` notiert wird, sondern stattdessen ein Attribut `name=`. Ein kompletter Anker sieht also so aus:
`<a name="Ankername">...</a>`

Den Ankernamen kannst du frei vergeben. Vergib keine zu langen Namen. Namen dürfen keine Leerzeichen und keine deutschen Umlaute enthalten. Das erste Zeichen muss ein Buchstabe sein. Danach sind auch Ziffern erlaubt. Benutze als Sonderzeichen im Namen höchstens den Unterstrich (_), den Bindestrich (-), den Doppelpunkt (:) oder den Punkt (.).

Was du zwischen `<a name="Ankername">` und `</a>` als Inhalt notierst, ist das Sprungziel für Verweise, die zu diesem Anker führen. Es ist durchaus erlaubt, einen leeren Anker zu notieren, also `<a name="Ankername"></a>`. Einige ältere Browser führen Verweise zu leeren Ankern jedoch nicht aus, weshalb es besser ist, den Anker immer um einen konkreten Inhalt zu setzen. Beachte dabei aber, dass das `a`-Element keine Block-Elemente als Inhalt haben darf. Wenn du also beispielsweise eine Überschrift als Anker definieren willst, was ja durchaus typisch ist, dann notiere die Elementverschachtelung immer in dieser Form:
`<h2><a name="Ankername">Text der Überschrift</a></h2>`

Um innerhalb einer Datei einen Verweis zu einem in der Datei vorhandenen Anker zu notieren, gilt folgendes Schema:
`<a href="#Ankername">Verweistext</a>`.
Das Verweisziel beginnt also mit einem Gatterzeichen #, unmittelbar gefolgt vom Ankernamen.

Wenn der Verweis zu einem Anker in einer anderen Datei führen soll, wird zuerst die Datei adressiert. Hinter dem Dateinamen folgt das Gatterzeichen `#` und dahinter der Ankername.

Obwohl HTML nicht zwischen Groß- und Kleinschreibung unterscheidet, ist es in jedem Fall empfehlenswert, Ankernamen bei Ankern und Verweisen dorthin genau gleich zu schreiben. Einige Browser führen den Verweis nämlich nicht aus, wenn der Ankername beim Anker und beim Verweis dorthin unterschiedliche Groß-/Kleinschreibung verwendet.

Wenn du **XHTML-konform** arbeiten willst, musst du in jedem Fall auf einheitliche Groß-/Kleinschreibung achten, da XHTML im Gegensatz zu HTML Groß-/Kleinschreibung streng unterscheidet.

Bei Verweisen innerhalb einer Datei erzeugt der Web-Browser, wenn die Datei in einer http-Umgebung angezeigt wird, keinen neuen Server-Zugriff, sofern er die Datei so vollständig in den Arbeitsspeicher geladen hat, dass er den Sprung ausführen kann.

Einige Browser, z.B. der Internet Explorer, kennen auch "intern reservierte" Ankernamen wie `top`. Wenn du also einen Verweis `<a href="#top">...</a>` notierst und kein Anker dieses Namens in der Datei existiert, springt der Browser beim Ausführen des Verweises an den Anfang der Seite.

Manche Browser haben Schwierigkeiten, zu Ankern zu springen, die innerhalb einer Tabelle notiert sind.

Es ist durchaus möglich, `a`-Elemente zu notieren, die sowohl ein `href`- als auch ein `name`-Attribut haben! Absichtlich selbstbezügliche Verweise kannst du beispielsweise so erzeugen:

```
<a name="hier" href="#hier">immer sch&ouml;n hierbleiben!</a>
```

Allgemeines zu projekt-externen Verweisen

Aus technischer Sicht stellen projekt-externe Verweise kein großes Problem dar. Zuvor wurde beschrieben, welche Möglichkeiten zur Referenzierung bestehen. Diese Möglichkeiten sind beim `href`-Attribut des `<a>-`Tags erlaubt.

Dennoch solltest du einiges mehr über projekt-externe Verweise wissen:

Im Normalfall darfst du ungefragt Verweise auf fremde Web-Angebote setzen. Du brauchst also keine E-Mail an den Anbieter mit Bitte um Genehmigung zu schreiben, wenn du auf sein Angebot einen Link setzen willst.

Jeder Anbieter, der mit seinem Web-Projekt online geht, muss sich im Klaren darüber sein, dass er Teil eines weltweiten Hypertext-Systems ist, in dem er nicht allein ist. Wenn er das nicht akzeptieren kann, ist er im Web fehl am Platz und hat das falsche Medium gewählt.

Es gibt jedoch Ausnahmen von der Regel. Wenn du beispielsweise selbst ein sehr stark frequentiertes Web-Angebot hast und auf dessen Einstiegsseite einen Verweis auf die Homepage eines kleinen, unbekannten Anbieters setzt, dann solltest du ihn vorher fragen. Der Grund: durch die vielen zu erwartenden Besucher, die über Ihren Verweis auf das fremde Angebot finden, wird dort plötzlich sehr viel **Traffic** (Besucherverkehr und Datenübertragung) erzeugt. Viele Anbieter haben bei ihrem Provider eine Volumenbegrenzung, und wenn diese überschritten wird, entstehen den Anbietern

unkalkulierbare Kosten. Auch könnte es sein, dass der fremde Server nicht sehr belastungsfähig ist und durch die vielen plötzlichen Besucher zusammenbrechen könnte.

Eine andere Ausnahme ist, wenn du den Verweis in einem negativen Kontext setzt. Wenn du also auf einen Anbieter verweist, nur weil du ihn auf deiner eigenen Seiten heftig kritisierst, dann ist es sicherer, sich dort eine Bestätigung einzuholen, dass ein solcher Verweis gebilligt wird. Juristisch sicher ist eine solche Bestätigung aber nur, wenn sie mit Unterschrift in dokumentenechter Form von einer dafür zuständigen Person geleistet wird.

Beispiele für projekt-externe Verweise

Die einzige Bedingung, die beim Anwender erfüllt sein muss, damit er projekt-externe Verweise ausführen kann, ist eine bestehende Internetverbindung.

Beispiel

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
       "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Besuchen Sie doch mal...</h1>

<p>
<a href="http://www.teamone.de/training/">ein paar Kurse &uuml;ber die
hier vermittelten Inhalte</a><br>
<a href="http://www.atomic-
eggs.com/selfspezial/guests/advguest.cgi?view">das <i>SELFSpezial</i>-
G&auml;stebuch</a><br>
<a href="http://babelfish.altavista.com/">den Babelfish</a><br>
<a href="ftp://ftp.uni-augsburg.de/">den FTP-Server der Universit&auml;t
Augsburg</a><br>
<a href="gopher://ftp.std.com/1">die ganze Welt auf einem Gopher-
Server</a><br>
<a href="telnet://locis.loc.gov/">die &ouml;ffentliche Telnet-Anwendung
der Library of Congress</a><br>
<a href="news:de.comm.infosystems.www.authoring.misc">die
deutschsprachige Newsgroup f&uuml;r HTML/CSS</a>
</p>

</body>
</html>
```

Erläuterung

Web-Adressen beginnen mit `http://`, manchmal auch mit `https://` (letzteres sind Server, bei denen die Datenübertragung von und zum Browser verschlüsselt stattfindet, z.B. bei Internet-Banking).

Andere Internet-Protokolle kannst du ebenfalls adressieren, beispielsweise FTP-Adressen mit `ftp://`, Adressen auf Gopher-Servern mit `gopher://` oder Telnet-Adressen mit `telnet://`. Auch Newsgroups im Usenet kannst du adressieren, nämlich mit `news:` (ohne die beiden sonst charakteristischen Schrägstriche). Bei anderen als `http`-Adressen kommt es auf den Web-Browser an, wie er damit umgeht. Die modernen Browser beherrschen meistens FTP und Gopher und stellen entsprechende Adressen in ihrem

Anzeigefenster dar. Bei Protokollen, die der Browser nicht unterstützt, versucht er, auf dem Rechner des Anwenders ein Programm auszuführen, das für das entsprechende Internet-Protokoll zuständig ist. Bei Telnet wird beispielsweise ein auf dem Rechner installierter Telnet-Client aufgerufen, und bei Verweisen auf Newsgroups ein Newsreader oder das Newsreader-Modul eines Mailprogramms. Bei Newsgroups muss jedoch ein News-Server im Newsreader des Anwenders eingerichtet sein, der die adressierte Newsgroup anbietet. Auch E-Mail-Verweise sind möglich.

Viele Adressen bestehen nur aus dem Namen einer "WWW-Domain", etwa `http://www.teamone.de/`. Trotzdem führt der Verweis auf eine konkrete HTML-Datei. Das liegt daran, dass es bei vielen Web-Servern einen so genannten Default-Dateinamen gibt - meistens `index.htm`, `index.html` oder `welcome.htm` bzw. `welcome.html`. Das Projekt muss natürlich auch eine entsprechende Datei besitzen. Im Verweis braucht die HTML-Datei aber nicht mit angegeben zu werden.

Viele solcher Adressen werden immer wieder ohne abschließenden Schrägstrich angegeben, etwa `http://www.teamone.de`. Es ist jedoch sauberer, wenn du noch den Schrägstrich dahinter setzt. Nur so kann der Web-Browser bereits am Verweis erkennen, dass es sich um ein Verzeichnis handelt, in dem eine Default-Datei steht, deren Namen der Web-Server zur Verfügung stellt. Es ist deshalb besser zu notieren: `http://www.teamone.de/`.

Noch wichtiger ist es, bei Unterverzeichnissen einen abschließenden Schrägstrich zu notieren. Zwar klappt es auch, wenn du eine Adresse wie `http://www.teamone.de/training` notierst. Doch dann findet unnötig viel Kommunikation zwischen Browser und Server statt, denn intern fordert der Browser vom Server im Beispiel erst mal eine Datei namens `training`, was den Server zunächst zu einer Fehlermeldung veranlasst, da diese Datei nicht existiert. Erst im zweiten Schritt wird erkannt, dass es sich um den Namen eines Verzeichnisses handelt. Notiere deshalb also **immer** Angaben wie `http://www.teamone.de/training/`, also mit abschließendem Schrägstrich.

Verweis-sensitive Grafiken definieren

Verweis-sensitive Grafiken sind Grafiken, in denen der Anwender mit der Maus auf ein Detail klicken kann. Daraufhin wird ein Verweis ausgeführt. Auf diese Weise kann der Anwender in einigen Fällen wesentlich intuitiver und schneller zu Information gelangen als durch lange verbale Verweislisten.

Beispiel

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
       "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Schnell zur Stadt oder Region Ihrer Wahl!</h1>

<map name="Landkarte">
<area shape="rect" coords="11,10,59,29" href="http://www.koblenz.de/"
alt="Koblenz">
```

```
<area shape="rect" coords="42,36,96,57" href="http://www.wiesbaden.de/"
alt="Wiesbaden">
<area shape="rect" coords="42,59,78,80" href="http://www.mainz.de/"
alt="Mainz">
<area shape="rect" coords="100,26,152,58"
href="http://www.frankfurt.de/" alt="Frankfurt">
<area shape="rect" coords="27,113,93,134" href="http://www.mannheim.de/"
alt="Mannheim">
<area shape="rect" coords="100,138,163,159"
href="http://www.heidelberg.de/" alt="Heidelberg">
<area shape="rect" coords="207,77,266,101"
href="http://www.wuerzburg.de/" alt="W&uuml;rzburg">
<area shape="rect" coords="282,62,344,85" href="http://www.bamberg.de/"
alt="Bamberg">
<area shape="rect" coords="255,132,316,150"
href="http://www.nuernberg.de/" alt="N&uuml;rnberg">
<area shape="rect" coords="78,182,132,200"
href="http://www.karlsruhe.de/" alt="Karlsruhe">
<area shape="rect" coords="142,169,200,193"
href="http://www.heilbronn.de/" alt="Heilbronn">
<area shape="rect" coords="140,209,198,230"
href="http://www.stuttgart.de/" alt="Stuttgart">
<area shape="rect" coords="187,263,222,281" href="http://www.ulm.de/"
alt="Ulm">
<area shape="rect" coords="249,278,304,297"
href="http://www.augsburg.de/" alt="Augsburg">
<area shape="poly" coords="48,311,105,248,96,210,75,205,38,234,8,310"
      href="http://www.baden-aktuell.de/" alt="Baden">
</map>

<p><img src="karte.gif" width="345" height="312" border="0" alt="Karte"
usemap="#Landkarte"></p>

</body>
</html>
```

Erläuterung

Mit `<map name="[Name]">` leitest du die Definition der verweis-sensitiven Flächen einer Grafik ein. Beim `name`-Attribut vergibst du einen Namen für die verweis-sensitive Grafik. Dieser Name muss nichts mit dem Dateinamen der Grafik zu tun haben. Es handelt sich vielmehr um einen Ankernamen, der die gleiche Bedeutung hat wie der Name in einem Anker innerhalb einer HTML-Datei. Vergib keine zu langen Namen. Namen dürfen keine Leerzeichen und keine deutschen Umlaute enthalten. Das erste Zeichen muss ein Buchstabe sein. Danach sind auch Ziffern erlaubt. Benutze als Sonderzeichen im Namen höchstens den Unterstrich (_), den Bindestrich (-), den Doppelpunkt (:) oder den Punkt (.).

Das `map`-Element kann an einer beliebigen Stelle innerhalb des Körpers einer HTML-Datei (also zwischen `<body>` und `</body>`) stehen. Es erzeugt selbst keine Bildschirmausgabe. Es empfiehlt sich jedoch, das Element an einer markanten, gesonderten Stelle innerhalb der Datei zu notieren, z.B. am Anfang oder am Ende des Dateikörpers.

Zwischen dem einleitenden `<map...>` und dem abschließenden `</map>` definierst du die verweis-sensitiven Flächen. Mit `<area...>` definierst du einzelne verweis-sensitive Flächen einer bestimmten Grafik, die du an einer anderen Stelle einbindest.

Mit `shape="rect"` bestimmst du eine viereckige Fläche, mit `shape="circle"` einen Kreis, und mit `shape="polygon"` kannst du ein beliebiges Vieleck als verweis-sensitiv definieren.

Bei `coords=` gibst du die Koordinaten der verweis-sensitiven Flächen an. Die Pixelangaben bedeuten absolute Werte innerhalb der Grafik, die verweis-sensitiv sein soll. Trenne alle Pixelwerte durch Kommata.

- Ein **Viereck** (shape="rect") definierst du mit den Koordinaten für x1,y1,x2,y2 wobei bedeuten:
  x1 = linke obere Ecke, Pixel von links
  y1 = linke obere Ecke, Pixel von oben
  x2 = rechte untere Ecke, Pixel von links
  y2 = rechte untere Ecke, Pixel von oben
- Einen **Kreis** (`shape="circle"`) definierst du mit den Koordinaten für x,y,r wobei bedeuten:
  x = Mittelpunkt, Pixel von links
  y = Mittelpunkt, Pixel von oben
  r = Radius in Pixel
- Ein **Polygon** (`shape="poly"`) definierst du mit den Koordinaten x1,y1,x2,y2 ... x_n,y_n
  wobei bedeuten:
  x = Pixel einer Ecke von links
  y = Pixel einer Ecke von oben
  Du kannst so viele Ecken definieren wie du willst. Von der letzten definierten Ecke musst du dir eine Linie zur ersten definierten Ecke hinzudenken. Diese schließt das Polygon.

Mit dem Attribut `href=` bestimmst du das Verweisziel, das aufgerufen werden soll, wenn der Anwender die verweis-sensitive Fläche anklickt. Dabei gelten die Regeln zum Referenzieren in HTML - es kann sich also um beliebige Verweisziele innerhalb oder außerhalb des eigenen Web-Projekts handeln. Wenn du eine Fläche explizit als nicht anklickbar ausweisen willst, kannst du dies mit dem Attribut `nohref` tun (ohne Wertzuweisung). Nötig ist dies allerdings nicht, denn Flächen, die nicht durch `href=` abgedeckt sind, sind automatisch nicht anklickbar.

Mit dem `alt`-Attribut notierst du Alternativtext für den Fall, dass die verweis-sensitive Fläche nicht angezeigt werden kann. Dieses Attribut ist ein **Pflichtattribut**, d.h. du musst es notieren, um gültiges HTML zu erzeugen.

Die Grafik, die verweis-sensitive Flächen haben soll, referenzierst du auf herkömmliche Weise mit Hilfe des `<img>`-Tags. Um die Grafik als verweis-sensitiv zu kennzeichnen, musst du das Attribut `usemap=` notieren. Dieses Attribut erwartet als Wertzuweisung einen URI, der zu der Stelle führt, an der das zugehörige `map`-Element notiert ist. Normalerweise ist dieses Element in der gleichen Datei notiert. Deshalb besteht die Zuweisung einfach in einem Gatterzeichen `#` und dem Namen des Ankers, der bei `<map name=>` definiert wurde.

Um die gewünschten Pixelkoordinaten für verweis-sensitive Flächen einer Grafik zu erhalten, kannst du beispielsweise ein Grafikprogramm benutzen, bei dem du mit der Maus in der angezeigten Grafik herumfahren kannst und dabei die genauen Pixelkoordinaten des Mauszeigers angezeigt bekommst.

Um beim Überfahren einer verweis-sensitiven Fläche mit der Maus ein kleines Fenster ("Tooltip"-Fenster) anzuzeigen, kannst du in den `<area>`-Tags jeweils das

Universalattribut `title="Mein angezeigter Text"` verwenden. Dies wird allerdings nicht von allen Browsern interpretiert.

Innerhalb eines jeden `<area>`-Tags ist auch das Attribut `tabindex=` erlaubt. Die Wirkung ist die gleiche wie bei der Tabulator-Reihenfolge und wird dort näher beschrieben. Ebenfalls erlaubt ist das Attribut `accesskey=`. Dies funktioniert genau wie bei Tastaturkürzlen und wird dort näher beschrieben.

Wenn du **XHTML-konform** arbeitest, musst du das `area`-Element als inhaltsleer kennzeichnen. Dazu notiere das alleinstehende Tag in der Form `<area... />`.

Im `<area>`-Tag kommen häufig auch JavaScript-Event-Handler zum Einsatz. Beim Aufruf von JavaScript-Funktionen mit Event-Handlern darf jedoch im `<area>`-Tag das `href`-Attribut nicht fehlen, sonst streikt der Netscape Navigator. Um das Problem zu lösen, kannst du eine Angabe notieren wie:
`<area shape="rect" coords="1,1,249,49" href="#" onclick="IhreFunktion()" alt="Kurze Beschreibung">`
Durch die Wertzuweisung `"#"` an das `href`-Attribut wird ein "leerer" Verweis erzeugt, und Netscape führt dann auch den Event-Handler (hier: `onClick=`) aus.

Verweis zu E-Mail-Adresse definieren

Du kannst auf jede beliebige E-Mail-Adresse im Internet-Format einen Verweis setzen. Eine Internet-gerechte E-Mail-Adresse erkennst du an dem Zeichen @ in der Mitte der Adresse. Wenn der Anwender auf den Verweis klickt, kann er eine E-Mail an den betreffenden Empfänger absetzen. Normalerweise benutzen Anbieter von WWW-Seiten diese Möglichkeit, um Besuchern die Möglichkeit zu bieten, eine E-Mail an die eigene Adresse senden, zum Beispiel wegen Feedback zum Angebot. Du kannst aber auch Verweise zu anderen E-Mail-Adressen anbieten.

Beispiel

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
       "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>E-Mail-Verweise definieren</title>
</head>
<body>

<h1>Impressum</h1>

<p>Verantwortlich f&uuml;r den Inhalt dieser Seiten ist:<br>
<a href="mailto:fritz.eierschale@lummerland.org">Fritz Eierschale,
fritz.eierschale@lummerland.org</a>
</p>

</body>
</html>
```

Erläuterung

Verweise auf E-Mail-Adressen funktionieren nach dem gleichen Schema wie alle anderen Verweise. Beim `href`-Attribut des einleitenden `<a>`-Tags wird das Verweisziel angegeben. Verweisziele vom Typ "E-Mail" beginnen immer mit `mailto:` (**ohne** `//` dahinter!).

Der Verweis ist bei Anwendern nur ausführbar, wenn der Web-Browser das Erstellen und Absenden von E-Mails unterstützt (z.B. Netscape), oder wenn bei solchen Verweisen automatisch ein E-Mail-Programm gestartet wird (dies ist z.B. beim MS Internet Explorer der Fall).

Es ist sinnvoll, im Verweistext die E-Mail-Adresse noch einmal explizit zu nennen, so wie im obigen Beispiel, damit Anwender, bei denen der E-Mail-Verweis nicht ausführbar ist, auf Wunsch separat eine E-Mail senden können.

Optionen bei E-Mail-Verweisen

Die hier beschriebenen Optionen sind nicht im HTML-Standard verankert, widersprechen diesem aber auch nicht unbedingt, solange die Kodierungsregeln für URIs eingehalten werden. Da sie von vielen Browsern interpretiert werden, sollen sie hier erwähnt werden.

Du kannst:

- E-Mail-Verweise an mehrere Empfänger senden, wahlweise cc (sichtbare Kopienempfänger, *cc = carbon copy*) oder bcc (unsichtbare Kopienempfänger, *bcc = blind carbon copy*),
- anstelle einer einfachen Mailadresse ein vollständiges Empfängerschema angeben von der Art:
  `Fritz Eierschale <fritz.eierschale@lummerland.org>`
- ein Subject vordefinieren, so dass beim Öffnen des E-Mail-Editors das Betreff-Feld bereits mit einem Text Ihrer Wahl ausgefüllt ist,
- einen Body-Inhalt vordefinieren, so dass beim Öffnen des E-Mail-Editors bereits Text im Nachrichtentext der E-Mail steht, z.B. eine Anrede,
- alle Optionen kombinieren.

Es besteht jedoch keinerlei Garantie, dass alle Möglichkeiten in allen Browsern und allen Kombinationen mit E-Mail-Programmen funktionieren.

Beispiel

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
       "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Tolle Mails</h1>

<p>Mail an einen Hauptempf&auml;nger und einen sichtbaren
Kopienempf&auml;nger:<br>
<a href="mailto:eierschale@irgend.wo?cc=heidi.bratze@vergiss.es">Mail
mit Kopie</a></p>
<p>Mail an einen Hauptempf&auml;nger und einen unsichtbaren
Kopienempf&auml;nger:<br>
<a href="mailto:eierschale@irgend.wo?bcc=heidi.bratze@vergiss.es">Mail
mit Geheimkopie</a></p>
<p>Mail mit vordefiniertem Subject (Betreff):<br>
<a href="mailto:eierschale@irgend.wo?subject=eine
Mail%20von%20deinen%20Web-Seiten">Mail mit Betreff</a></p>
<p>Mail mit vordefiniertem Body (Nachrichtentext):<br>
```

```
<a
href="mailto:eierschale@irgend.wo?body=Hallo%20Fritz,%0D%0A%0D%0Aich%20w
ollte%20nur%20sagen,%20da%DF ">Mail mit Body</a></p>
<p>Mail mit kombinierten Optionen:<br>
<a
href="mailto:eierschale@irgend.wo?cc=heidi.bratze@vergiss.es&amp;subject
=Hallo%20Fritz,%20hallo%20Heidi">cc und Subject</a></p>
<p>Mail an zwei Hauptempf&auml;nger:<br>
<a href="mailto:eierschale@irgend.wo,%20heidi.bratze@vergiss.es">Mail an
zwei</a></p>
<p>Mail mit vollst&auml;ndigem Adressierungsschema:<br>
<a href="mailto:Fritz%20Eierschale%20&lt;eierschale@irgend.wo&gt;">Mail
mit Adressierungsschema</a></p>

</body>
</html>
```

Erläuterung

Die Optionen werden in dem Wert notiert, der dem `href`-Attribut zugewiesen wird. Es gibt also keine speziellen HTML-Attribute für diese Optionen, und deshalb ist die Handhabung auch etwas umständlich. Zuerst wird wie üblich der Empfänger notiert. In den obigen Beispielen (mit Ausnahme des letzten) geschieht das jeweils durch `mailto:eierschale@irgend.wo`. Dahinter wird ein Fragezeichen `?` notiert. Das ist in der URI-Syntax das übliche Zeichen, um Parameter an eine aufgerufene Adresse zu übergeben. Hinter dem Fragezeichen folgen die Parameter. Jede Option ist so ein Parameter und besteht aus einem Namen, einem Istgleichzeichen und einem zugewiesenen Wert. Als Optionsnamen sind erlaubt:

`cc` (sichtbarer Kopienempfänger),
`bcc` (unsichtbarer Kopienempfänger),
`subject` (Betreff) und
`body` (Nachrichtentext).

Ein Konstrukt wie `cc=heidi.bratze@vergiss.es` ist also ein vollständiger Parameter und bedeutet: "sichtbare Kopie an heidi.bratze@vergiss.es".

Bei den Wertzuweisungen an die Parameter können Zeichen vorkommen, die nicht zu einer gültigen URI gehören. Damit die URI gültig bleibt (andernfalls wäre das Dokument auch kein gültiges HTML mehr), müssen diverse Zeichen maskiert werden. Die Maskierung besteht darin, ein Prozentzeichen `%` zu notieren, gefolgt von dem hexadezimal ausgedrückten Zeichenwert des gewünschten Zeichens.

Die folgende Tabelle listet Zeichen auf, die maskiert werden müssen, weil sie innerhalb von URIs nicht vorkommen dürfen oder eine bestimmte Bedeutung haben. Links steht das Zeichen, rechts die Zeichenkette, mit der du das Zeichen maskieren musst:

Zeichen	Zeichenkette für Maskierung
[neue Zeile]	%0A
[Wagenrücklauf]	%0D
[Leerzeichen]	%20
!	%21
#	%23
%	%25
*	%2A

/	%2F
?	%3F

Ferner musst du alle Zeichen maskieren, die oberhalb des ASCII-Zeichensatzes liegen, also z.B. deutsche Umlaute und scharfes S. Die folgende Tabelle listet die wichtigsten Zeichen auf und ihre Maskierung auf:

Zeichen	Zeichenkette für Maskierung
Ä	%C4
ä	%E4
Ö	%D6
ö	%F6
Ü	%DC
ü	%FC
ß	%DF

Wenn du mehrere Optionen kombinieren willst, musst du die Optionen durch ein kaufmännisches Und (&) voneinander trennen. Nun darf ein solches Zeichen in HTML - auch nicht bei einer Wertzuweisung an ein Attribut - nicht unmaskiert vorkommen. Deshalb solltest du es wie in HTML üblich mit `&` (=&) maskieren. Leider gibt es einige ältere Browser (z.B. Netscape 3.x, die damit nicht klarkommen).

Die folgende Zeichenkette:
`cc=heidi.bratze@vergiss.es&subject=Hallo%20Fritz,%20hallo%20Heidi`
kombiniert im obigen Beispiel also die Optionen für `cc` und `subject`. Um weitere Optionen hinzufügen, notiere ein weiteres `&` (=&) und dahinter eine weitere Option, bestehend aus Optionsname, Istgleichzeichen und zugewiesenem Wert.

Ein Schema, wie du es vermutlich aus deiner E-Mail-Korrespondenz kennst, lautet beispielsweise:
`Fritz Eierschale <eierschale@irgend.wo>`
In einem E-Mail-Verweis muss die Zuweisung an das `href`-Attribut jedoch so aussehen:
`mailto:Fritz%20Eierschale%20<eierschale@irgend.wo>`
Auch in diesem Fall musst du also alle weiter oben erwähnten Zeichen maskieren. Außerdem musst du die beiden Zeichen `<` und `>` mit ihren HTML-gerechten Maskierungen `<` bzw. `>` umschreiben.

Download-Verweise

Es gibt keine spezifische Notation in HTML, um Dateien beim Anklicken zum Downloaden anzubieten. Es gibt lediglich Dateitypen, die (fast) jeder Web-Browser so interpretiert, dass er dem Anwender anbietet, die Datei downzuloaden. Das bekannteste Dateiformat dafür ist heute das ZIP-Format (*.zip). ZIP-Dateien sind Archivdateien, die mehrere andere Dateien enthalten können, sogar ganze Verzeichnisstrukturen. Die enthaltenen Dateien werden außerdem komprimiert. Der Anwender muss die ZIP-Datei nach dem Download mit einem geeigneten Programm entpacken ("Unzip"-Programm).

Beispiel

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
       "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Download-Ecke</h1>

<p><a href="wavetext.zip"><b>WaveText</b></a><br>
Ein Java-Applet f&uuml;r Textanimationen (wavetext.zip, 2.106 Bytes)</p>

</body>
</html>
```

Erläuterung

Verweise auf typische Download-Dateien wie ZIP-Dateien unterscheiden sich nicht von anderen Verweisen. Das Verweisziel ist die Datei, die zum Download angeboten wird. Wenn die Datei zum eigenen Projekt gehört, gelten die gleichen Regeln wie bei projekt-internen Verweisen, bei entfernten Dateien die Regeln von projekt-externen Verweisen.

Wenn du Dateien anbietest, die nur für bestimmte Rechnerumgebungen gedacht sind, kannst du natürlich auch Formate verwenden, die speziell für diese Umgebungen gedacht sind. Für DOS/Windows-Umgebungen können das beispielsweise selbstentpackende EXE-Archive sein, oder für Macintosh HQX-Archive. Wenn der Web-Browser mit der Dateiendung nichts anfangen kann, sollte er den Anwender im Dialog entscheiden lassen, was er mit der Datei tun möchte - dabei sollte auch die Download-Möglichkeit angeboten werden.

Verweise zu beliebigen Dateien

Du kannst auf jede beliebige Datei einen Verweis setzen. Es kann sich um Audio-Dateien, Tabellenkalkulations-Dateien, CAD-Dateien, Video-Dateien, Grafikdateien, Textverarbeitungs-Dateien, Programmdateien, Datenbankdateien handeln - was du willst. Du kannst auch jede beliebige Datei in dein eigenes Web-Projekt mit auf den Internet-Server laden und einen Verweis darauf setzen. Aus Sicht von HTML ist das kein Problem. Das Problem besteht darin, was der Web-Browser des Anwenders mit den Dateien anfangen kann, bzw. wie er ihren Inhalt korrekt anzeigen oder abspielen lassen kann.

Beispiel

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
       "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Stuff</h1>

<p><a href="buch.url"><b>buch.url</b></a><br>
Eine Favoritendatei, wie der Internet Explorer sie anlegt</p>
```

```
<p><a href="fritz.vcf"><b>fritz.vcf</b></a><br>
Eine Visitenkartendatei (Adressenaustauschformat)</p>
<p><a href="einsundeins.xls"><b>einsundeins.xls</b></a><br>
Eine Excel-Datei</p>

</body>
</html>
```

Erläuterung

Verweise auf beliebige Dateien unterscheiden sich nicht von anderen Verweisen. Das Verweisziel ist die gewünschte Datei. Wenn die Datei zum eigenen Projekt gehört, gelten die gleichen Regeln wie bei projekt-internen Verweisen, bei entfernten Dateien die Regeln von projekt-externen Verweisen.

Dateitypen wie reine Textdateien (*.txt) kann der Web-Browser selbst anzeigen.

Moderne Web-Browser haben eine Plugin-Schnittstelle. Wenn der Anwender ein Plugin zur Darstellung des Dateityps besitzt, kann der Browser die Datei mit Hilfe des Plugins selbst anzeigen bzw. abspielen.

Wenn der Anwender ein Programm besitzt, das den Dateityp verarbeiten kann, und dem Web-Browser ist die Verknüpfung zwischen Dateien mit der Endung des Verweisziels und einem Programm bekannt, das solche Dateien verarbeitet, dann kann der Browser das Programm starten. Wenn das Betriebssystem, der Web-Browser und das andere Programm den dynamischen Datenaustausch zwischen Programmen erlauben, kann das Anzeigefenster des Fremdprogramms in das Browser-Fenster eingebettet werden. Es gibt jedoch keine Möglichkeit, solche Dinge in irgendeiner Weise als Web-Autor zu beeinflussen.

Wenn der Browser mit dem Dateityp gar nichts anfangen kann, sollte er dem Anwender einen Dialog anbieten, um zu entscheiden, was mit der Datei geschehen soll. Der Anwender sollte die Datei z.B. downloaden können.

Mime-Type des Verweisziels angeben

Um dem Browser die Aufgabe zu erleichtern, um welche Art von Datei es sich handelt, kannst du den Mime-Type der Datei angeben, auf den du verweist.

Mime steht für **M**ultipurpose **I**nternet **M**ail **E**xtensions. Aus dem Namen geht hervor, dass das, was da spezifiert wird, ursprünglich für E-Mails gedacht war - und zwar für E-Mails mit Attachments (englisch für *Anhang*). Solche so genannten Multipart-Mails enthalten die gesamten zu übertragenden Daten in einer Datei. Innerhalb der Datei musste eine Konvention gefunden werden, wie die einzelnen Teile (z.B. Text der Mail und angehängte ZIP-Datei) voneinander zu trennen seien. Dabei wurde auch ein Schema entwickelt, das der interpretierenden Software mitteilt, um welchen Datentyp es sich bei dem jeweils nächsten Teil der Mail handelt.

Das Schema erwies sich nicht nur für E-Mails als nützlich. Fast immer, wenn entfernte Programme (z.B. Web-Browser und Web-Server) wegen einer bevorstehenden Datenübertragung miteinander kommunizieren, geht es auch um die Art der zu übertragenden Daten. Dabei hat sich im gesamten Internet das Schema der Mime-Typen durchgesetzt. Eine Übersicht verfügbarer Mime-Typen findest du im Anhang.

Beispiel

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
       "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Stuff mit Typ</h1>

<p><a href="einsundeins.xls"
type="application/msexcel"><b>einsundeins.xls</b></a><br>
Eine Excel-Datei</p>

</body>
</html>
```

Erläuterung

Mit dem zusätzlichen Attribut `type=` kannst du den Mime-Type der Zieldatei bestimmen, im obigen Beispiel `application/msexcel` für Excel-Dateien.

Formulare

HTML stellt die Möglichkeit zur Verfügung, Formulare zu erstellen. In Formularen kann der Anwender Eingabefelder ausfüllen, in mehrzeiligen Textfeldern Text eingeben, aus Listen Einträge auswählen usw. Wenn das Formular fertig ausgefüllt ist, kann der Anwender auf einen Button klicken, um das Formular abzusenden.

Dazu gibst du beim Erstellen eines Formulars an, was mit den Daten des ausgefüllten Formulars passieren soll. Du kannst dir die ausgefüllten Daten beispielsweise per E-Mail zuschicken lassen oder von einem CGI-Programm auf dem Server-Rechner weiterverarbeiten lassen.

Formulare können sehr unterschiedliche Aufgaben haben. So werden sie zum Beispiel eingesetzt:

- um bestimmte, gleichartig strukturierte Auskünfte von Anwendern einzuholen,
- um Anwendern das Suchen in Datenbeständen zu ermöglichen,
- um Anwendern die Möglichkeit zu geben, selbst Daten für einen Datenbestand beizusteuern,
- um dem Anwender die Möglichkeit individueller Interaktion zu bieten, etwa um aus einer Produktpalette etwas Bestimmtes zu bestellen.

Ein Software-Hersteller könnte z.B. ein Formular zur Verfügung stellen, in dem der Anwender angeben kann, welche Produkte der Firma er besitzt, wie er Kenntnis von den Produkten erhalten hat, welchen Beruf er ausübt, auf welchem Rechnertyp die Software bei ihm läuft usw. Auf diese Weise könnte er gleichartig strukturiertes und daher gut vergleichbares Feedback von Anwendern einholen.

Viele Suchdienste im Internet bieten dem aufrufenden Web-Browser Eingabe-Formulare an, in denen der Anwender seinen Suchwunsch spezifizieren kann. Ohne solche Formulare wäre das Durchsuchen gar nicht möglich. Die meisten Suchdienste bieten darüber hinaus auch die Möglichkeit an, eigene Internet-Adressen in die Datenbank einzuspeisen. Das Einholen der dazu nötigen Information geschieht ebenfalls mit Hilfe von Formularen.

Immer zahlreicher werden auch die Online-Shops im Internet. Egal ob Tickets, Pizza oder Unterwäsche - um solche Bestell-Services zu realisieren, sind Formulare erforderlich, in denen der Anwender seine Bestellwünsche genau angeben kann.

Formularbereich definieren

Du kannst an einer beliebigen Stelle innerhalb des Dateikörpers einer HTML-Datei ein Formular definieren.

Beispiel 1

```
<form action="http://www.ihr-guter-name.de/cgi-bin/feedback.pl"
method="get">
<!-- hier folgen die Formularelemente -->
</form>
```

Beispiel 2

```
<form action="mailto:eierschale@irgend.wo" method="post"
enctype="text/plain">
<!-- hier folgen die Formularelemente -->
</form>
```

Erläuterung

Mit `<form>...</form>` definierst du ein Formular (*form* = *Formular*). Alles, was zwischen dem einleitenden `<form>`-Tag und dem abschließenden Tag `</form>` steht, gehört zum Formular. Das sind hauptsächlich Elemente des Formulars wie Eingabefelder, Auswahllisten oder Buttons. Um die Elemente zu platzieren und zu beschriften, kannst du dazwischen aber auch andere HTML-Elemente notieren. Dabei musst du allerdings folgendes beachten: nach der HTML-Variante "Strict" darfst du innerhalb eines Formulars nur Block-Elemente (und Script-Bereiche) notieren, also etwa Überschriften, Textabsätze, allgemeine Bereiche oder Tabellen. In der HTML-Variante "Transitional" ist es dagegen außerdem erlaubt, zwischen `<form>` und `</form>` auch gemischten Inhalt aus Text und Inline-Elementen zu notieren. Um die etwas lästige und an dieser Stelle nicht ganz nachvollziehbare Restriktion in der "Strict"-Variante zu umgehen, kannst du dir damit behelfen, dass du das Formular folgendermaßen strukturierst:
`<form><div> <!-- Formularinhalt --> </div></form>`.
Innerhalb des `div`-Elements, das da innerhalb des Formulars notiert ist, hast du auch in der "Strict"-Variante die Freiheiten der "Transitional"-Variante.

Im einleitenden `<form>`-Tag gibst du mit dem Pflichtattribut `action=` an, was mit den Formulardaten passieren soll, wenn der Anwender das Formular absendet (*action* = *Aktion*). Die Wertzuweisung an `action=` kann beispielsweise eine E-Mail-Adresse (normalerweise deine eigene) mit vorangestelltem `mailto:` sein - so wie im zweiten der obigen Beispiele (*mailto* = *sende an*). Dann werden die ausgefüllten Formulardaten an

diese E-Mail-Adresse geschickt, sofern das möglich ist.
Oder du rufst ein Programm auf dem Server-Rechner, meistens ein CGI-Script, auf, das die Daten weiterverarbeitet - so wie im ersten der obigen Beispiele.
Du kannst bei `action=` auch eine HTML-Datei angeben. Diese wird bei Absenden des Formulars aufgerufen und kann die Formulardaten z.B. mit JavaScript weiterverarbeiten. Das ist beispielsweise für mehrseitige Formulare interessant. Berücksichtige dabei aber, dass JavaScript nur dann Zugriff auf Daten hat, wenn die Methode `get` verwendet wurde. Bei einigen Browsern, z.B. bei Opera, funktioniert das Übergeben von Formulardaten zwischen HTML-Dateien auch nur in HTTP-Umgebung, also nicht lokal ohne Serverkommunikation.

Bei der Wertzuweisung an `action=` gelten die Regeln zum Referenzieren in HTML. Das Referenzieren mit relativen oder absoluten Pfadangaben ist also ebenso möglich wie das oben in Beispiel 1 gezeigte Referenzieren mit einem absoluten URI, z.B.:
`<form action="/cgi-bin/auswert.pl">`
`<form action="../../cgi-bin/suche.cgi">`.

Eine weiteres wichtiges Attribut bei der Formulardefinition ist das Attribut `method=`. Dabei bestimmst du, nach welcher HTTP-Übertragungsmethode die Formulardaten an ihr Ziel gelangen. Dabei gibt es zwei mögliche Wertzuweisungen:

- Wenn du `method="get"` wählst (diese Angabe ist übrigens nicht zwingend erforderlich, da `get` als Default-Einstellung definiert ist), werden die Daten des ausgefüllten Formulars als Parameter an die Aufrufadresse angehängt. Die Anfrage, die beim Server eintrifft, sieht dann beispielsweise so aus:
  `http://www.ihr-guter-name.de/cgi-bin/feedback.cgi?AnwenderName=Stefan+M%FCnz&AnwenderMail=selfhtml@teamone.de&Text=Das+ist+ein+kleiner+Text`.
  Das verarbeitende Script kann diese als Parameter übergebene Zeichenkette auslesen und für die Datenverarbeitung auseinander dröseln.
- Wenn du `method="post"` wählst, werden die Daten des ausgefüllten Formulars vom Web-Server über den Standardeingabekanal zur Verfügung gestellt, und ein auswertendes CGI-Script muss die ankommenden Daten behandeln wie eine Benutzereingabe, die auf der Kommandozeile gemacht wurde (*post = verschicken*). Bei CGI-Scripts musst du diese Methode auf jeden Fall dann verwenden, wenn das Script die `POST`-Methode verarbeitet, und wenn die Formulardaten zu umfangreich für die `GET`-Methode sind.

Das W3-Konsortium empfiehlt, die Methode `get` dann zu wählen, wenn das auswertende Programm die Daten nur als Input benötigt (z.B. für eine Suche), während die Methode `post` für Fälle empfohlen wird, in denen das auswertende Programm "Seiteneffekte" wie das Einspeisen der Daten in eine Datenbank ausführt.

Wenn du dir die Formulardaten per E-Mail zuschicken lässt, wie im zweiten obigen Beispiel gezeigt, dann benutze immer `method="post"`. Außerdem solltest du bei E-Mail-Empfang von Formulardaten im einleitenden `<form>`-Tag `enctype="text/plain"` mit angeben. Denn Formulardaten sind normalerweise nach einem Schema kodiert, das für Menschen keine Freude zu lesen ist. Mit der genannten Angabe erhaltest du zumindest von Anwendern, die dein Formular mit einem modernen Web-Browser ausfüllen, ordentlich formatierte E-Mails.

Bei E-Mail-Formularen besteht keine Garantie auf Erfolg. Es hängt vom Browser und

anderen Einstellungen auf dem Rechner des Anwenders ab, ob der Formularversand klappt. E-Mail-Formulare gelten deshalb mittlerweile als unsauber, zumal es Alternativen gibt.

Zielfenster für Server-Antwort

Wenn du mit Frames arbeitest und in einem Frame-Fenster ein Formular hast, nach dessen Absenden die Server-Antwort bzw. die Antwort eines CGI-Scripts in einem anderen Frame-Fenster angezeigt werden soll, kannst du das gewünschte Zielfenster angeben.

Beispiel

```
<form action="/cgi-bin/auswert.pl" method="get" target="Daten">
<!-- hier folgen die Formularelemente -->
</form>
```

Erläuterung

Mit dem Attribut `target=` kannst du im einleitenden `<form>`-Tag den Namen des Frame-Fensters angeben, in dem die Server-Antwort ausgegeben werden soll. Es muss sich entweder um einen Fensternamen handeln, der für ein Frame-Fenster dem `name`-Attribut im `<frame>`-Tag vergeben wurde, oder um einen der folgenden reservierten Fensternamen:
`_self`, um die Server-Antwort im aktuellen Fenster auszugeben,
`_parent`, um für die Server-Antwort bei verschachtelten Framesets das aktuelle Frameset zu sprengen,
`_top`, um für die Server-Antwort bei verschachtelten Framesets alle Framesets zu sprengen.

Das `target`-Attribut ist zwar nicht als *deprecated* (missbilligt) gekennzeichnet, doch um es einzusetzen, musst du die HTML-Variante "Transitional" verwenden. Der Grund ist, dass dieses Attribut vorwiegend für Verweise bei Verwendung von Frames gedacht ist und Frames eine eigene HTML-Variante haben, die von der Einstufung her der Variante "Transitional" entspricht (auf gut Deutsch: nicht der "reinen Lehre" entspricht).

Zu verarbeitende Zeichensätze

Du kannst angeben, nach welchen Zeichensätzen eingegebene Formulardaten interpretiert werden sollen. Dabei kannst du einen oder mehrere Zeichensätze angeben.

Beispiel

```
<form action="/cgi-bin/auswert.pl" method="get" accept-charset="ISO-
8859-1, ISO-8859-2">
<!-- hier folgen die Formularelemente -->
</form>
```

Erläuterung

Mit dem Attribut `accept-charset=` kannst du Zeichensätze angeben. Trenne mehrere Angaben durch Kommata und/oder Leerzeichen.

Formularelemente

Einzeilige Eingabefelder definieren

Einzeilige Eingabefelder dienen zur Aufnahme von einem oder wenigen Wörtern oder einer Zahl.

Beispiel

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
       "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Formular f&uuml;r Namenseingabe</h1>

<form action="input_text.htm">
<p>Vorname:<br>
<input name="vorname" type="text" size="30" maxlength="30">
</p>
<p>Zuname:<br>
<input name="zuname" type="text" size="30" maxlength="40">
</p>
</form>

</body>
</html>
```

Erläuterung

`<input>` definiert ein einzeiliges Eingabefeld (*input = Eingabe*). Der Vollständigkeit halber solltest du die Angabe `type="text"` dazusetzen.

Jedes Eingabefeld sollte einen internen Bezeichnernamen erhalten, und zwar mit der Angabe `name=`. Der vergebene Name wird bei CGI-Scripts benötigt, um auf die Daten des Eingabefeldes zugreifen zu können. Bei E-Mail-Formularen taucht der Name als Feldname auf. Und auch bei dem Formularfeldzugriff mit JavaScript ist der Name von Bedeutung. Der Name sollte nicht zu lang sein und darf keine Leerzeichen, Sonderzeichen oder deutsche Umlaute enthalten. Das erste Zeichen muss ein Buchstabe sein. Danach sind auch Ziffern erlaubt. Benutze als Sonderzeichen im Namen höchstens den Unterstrich (_), den Bindestrich (-), den Doppelpunkt (:) oder den Punkt (.). Im Hinblick auf JavaScript darf der Name sogar nur Buchstaben, Ziffern und den Unterstrich (_) enthalten. Groß- und Kleinschreibung werden bei den meisten Programmiersprachen ebenfalls unterschieden.

Ferner solltest du bei einzeiligen Eingabefeldern immer die Anzeigelänge in Zeichen mit `size=` sowie die interne Feldlänge in Zeichen `maxlength=` bestimmen. Beide Angaben bedeuten die Anzahl Zeichen (*size = Größe, maxlength = maximal length = maximale Länge*). Wenn die interne Feldlänge `maxlength=` größer ist als die angezeigte Feldlänge

`size=` (wie im zweiten Eingabefeld des obigen Beispiels), dann wird bei längeren Eingaben automatisch gescrollt (im Beispiel also ab dem 31. eingegebenen Zeichen).

Um Beschriftung und Eingabefelder ordentlich zu positionieren, empfiehlt sich übrigens der Einsatz einer "blinden" Tabelle.

Beispiel

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
       "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Formular f&uuml;r Namenseingabe</h1>

<form action="input_text_tabelle.htm">
<table border="0" cellpadding="0" cellspacing="4">
 <tr>
  <td align="right">Vorname:</td>
  <td><input name="vorname" type="text" size="30" maxlength="30"></td>
 </tr><tr>
  <td align="right">Zuname:</td>
  <td><input name="zuname" type="text" size="30" maxlength="40"></td>
 </tr>
</table>
</form>

</body>
</html>
```

Erläuterung

Das Beispiel zeigt das gleiche Formular wie im Beispiel weiter oben. Diesmal stehen jedoch Beschriftung und zugehöriges Formularfeld in je einer Tabellenzeile sauber nebeneinander. Die Beschriftungstexte werden dabei rechtsbündig ausgerichtet und erscheinen dadurch bündig zu den Eingabefeldern hin orientiert.

Wichtig ist, dass das `form`-Element außerhalb der Tabelle steht, oder andersherum, dass die Tabelle als Element innerhalb des Formulars notiert wird.

Das `<input>`-Tag ist ein so genanntes Standalone-Tag. Wenn du **XHTML-konform** arbeitest, musst du das `input`-Element als inhaltsleer kennzeichnen. Dazu notierst du das alleinstehende Tag in der Form `<input... />`.

Mit Hilfe von JavaScript kannst du die Eingaben von Anwendern vor dem Absenden des Formulars kontrollieren. So kannst du zum Beispiel erzwingen, dass in einem Eingabefeld eine Zahl in einem bestimmten Wertebereich eingegeben wird usw..

Textvorbelegung bei einzeiligen Eingabefeldern

Du kannst ein einzeiliges Eingabefeld mit einem Inhalt vorbelegen.

Beispiel

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
       "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Einfach &uuml;berschreiben...</h1>

<form action="input_text.htm">
<p>Vorname:<br>
<input name="vorname" type="text" size="30" maxlength="30"
value="Michaela">
</p>
<p>Zuname:<br>
<input name="zuname" type="text" size="30" maxlength="40"
value="Mustermann">
</p>
</form>

</body>
</html>
```

Erläuterung

Mit dem zusätzlichen Attribut `value=` kannst du einen Text bestimmen, mit dem das Feld vorbelegt wird (*value = Wert*).

Eingabefelder für Passwörter

Für die Eingabe von Geheimnummern, Passwörtern usw. gibt es einen speziellen Typ von Eingabefeld. Die eingegebenen Zeichen werden dabei durch Platzhalter (meistens Sternchen) dargestellt, so dass Personen im Raum des Anwenders nicht zufällig das eingegebene Passwort mitlesen können.

Beispiel

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
       "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Nur nicht auf die Tastatur gucken lassen ... :-)</h1>

<form action="input_password.htm">
<p>Kennwort:<br>
<input name="kennwort" type="password" size="12" maxlength="12">
</p>
</form>

</body>
</html>
```

Erläuterung

Eingabefelder für Passwörter werden mit `type="password"` im `<input>`-Tag definiert.

Passwörter werden beim normalen HTTP-Protokoll trotz der verdeckten Eingabe im

Klartext über das Internet übertragen. Weise Anwender in ernsthaften Zusammenhängen auf diese Tatsache bitte explizit hin. Für eine verschlüsselte Kommunikation zwischen Browser und Server gibt es das HTTPS-Protokoll, das der Web-Server unterstützen muss. Frage gegebenenfalls deinen Provider danach.

Mehrzeilige Eingabebereiche definieren

Mehrzeilige Eingabefelder dienen zur Aufnahme von Kommentaren, Nachrichten usw.

Beispiel

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
       "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Ganz spontan</h1>

<form action="textarea.htm">
<p>Welche HTML-Elemente fallen Ihnen ein, und was bewirken sie:<br>
<textarea name="user_eingabe" cols="50" rows="10"></textarea>
</p>
</form>

</body>
</html>
```

Erläuterung

`<textarea ...>` leitet ein mehrzeiliges Eingabefeld ein (*textarea* = *Textbereich*). Jedes mehrzeilige Eingabefeld sollte ebenso wie ein einzeiliges Eingabefeld einen internen Bezeichnernamen erhalten, und zwar mit der Angabe `name=`.

Pflicht ist die Angabe zur Höhe und Breite des Eingabebereichs. Das Attribut `rows=` bestimmt die Anzahl der angezeigten Zeilen (*rows* = *Zeilen*) und damit die Höhe, während `cols=` die Anzahl der angezeigten Spalten (*cols* = *columns* = *Spalten*) festlegt. "Spalten" bedeutet dabei die Anzahl Zeichen (pro Zeile).

Mit `</textarea>` schließt du das mehrzeilige Eingabefeld ab. Das `</textarea>`-Tag ist nötig und darf nicht weggelassen werden.

Die Attribute `rows=` und `cols=` bestimmen lediglich die Anzeigegröße des Eingabebereichs, nicht die Länge des erlaubten Textes. Die ist theoretisch unbegrenzt. Web-Browser statten die mehrzeiligen Eingabefelder bei der Anzeige üblicherweise mit Scrollbalken aus, so dass der Anwender bei längeren Eingaben scrollen kann.

Textvorbelegung bei mehrzeiligen Eingabebereichen

Du kannst ein mehrzeiliges Eingabefeld mit Inhalt vorbelegen.

Beispiel

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
       "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Schreiben Sie eine Geschichte!</h1>

<form action="textarea.htm">
<p>Hier der Anfang der Geschichte:<br>
<textarea name="user_eingabe" cols="50" rows="10">
Es war dunkel, feucht und neblig, ein richtiger Novemberabend. Bernie
hatte alles
so weit vergessen, was geschehen war, und stapfte gedankenverloren und
irgendwie
leer durch die Stra&szlig;e. Da pl&ouml;tzlich ...
</textarea></p>
</form>

</body>
</html>
```

Erläuterung

Um mehrzeilige Eingabefelder mit Text vorzubelegen, notiere den gewünschten Text einfach als Elementinhalt zwischen dem einleitenden `<textarea>`-Tag und vor dem abschließenden `</textarea>`.

Eingabefelder und Eingabebereiche auf "nur lesen" setzen

Du kannst erzwingen, dass ein Eingabefeld kein Eingabefeld mehr ist, sondern eigentlich nur ein Ausgabefeld, und ein Eingabebereich nur ein Ausgabebereich. Das kann beispielsweise interessant sein, wenn du in JavaScript-ermittelte Werte in einem Formularfeld ausgeben möchtest, oder wenn du Felder mit einem Wert vorbelegen möchtest, den der Anwender aber nicht ändern können soll.

Beispiel

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
       "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Kein Rabatt und keine Gnade</h1>

<form action="input_text_value.htm">
<p>Preis:<br>
<input name="vorname" type="text" size="8" value="&euro; 699.-"
readonly>
</p>
<p>Lizenzvereinbarung:<br>
<textarea name="lizenz" cols="50" rows="10" readonly>
Mit dem Absenden des Formulars stimme ich zu, dass ich diese
Lizenzvereinbarung gelesen habe usw.
</textarea>
</p>
</form>

</body>
```

```
</html>
```

Erläuterung

Mit dem Attribut `readonly` kannst du ein einzeiliges Eingabefeld oder einen mehrzeiligen Eingabereich auf "nur lesen" setzen.

Bei älteren Browsern funktioniert diese Angabe nicht, und der Feldeintrag ist für den Anwender überschreibbar!

Wenn du XHTML-konform arbeiten willst, musst du das Attribut in der Form `readonly="readonly"` notieren.

Auswahllisten

Du kannst dem Anwender eine Liste mit festen Einträgen anbieten, aus der er einen Eintrag auswählen kann. Der Text des ausgewählten Eintrags wird übertragen, wenn der Anwender das Formular abschickt.

Beispiel

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
       "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>W&auml;hlen Sie Ihren Favoriten!</h1>

<form action="select.htm">
<p>
<select name="top5" size="3">
<option>Heino</option>
<option>Michael Jackson</option>
<option>Tom Waits</option>
<option>Nina Hagen</option>
<option>Marianne Rosenberg</option>
</select>
</p>
</form>

</body>
</html>
```

Erläuterung

`<select ...>` leitet eine Auswahlliste ein. Jede Auswahlliste sollte einen internen Bezeichnernamen erhalten, und zwar mit dem Attribut `name=`. Der Name sollte nicht zu lang sein und darf keine Leerzeichen, Sonderzeichen oder deutsche Umlaute enthalten. Das erste Zeichen muss ein Buchstabe sein. Danach sind auch Ziffern erlaubt. Benutze als Sonderzeichen im Namen höchstens den Unterstrich (_), den Bindestrich (-), den Doppelpunkt (:) oder den Punkt (.).

Im Hinblick auf JavaScript darf der Name sogar nur Buchstaben, Ziffern und den Unterstrich (_) enthalten. Groß- und Kleinschreibung werden bei den meisten

Programmiersprachen ebenfalls unterschieden.

Mit dem Attribut `size=` bestimmst du die Anzeigegröße der Liste, d.h. wie viele Einträge angezeigt werden sollen. Wenn die Liste mehr Einträge enthält als angezeigt werden, kann der Anwender in der Liste scrollen. Wenn du `size="1"` angibst, definierst du eine so genannte "Dropdown-Liste".

Mit `<option>...</option>` definierst du zwischen dem einleitenden `<select>`-Tag und dem Abschluss-Tag `</select>` jeweils einen Eintrag der Auswahlliste. Hinter `<option>` muss der Text des Listeneintrags stehen. Du kannst so viele Listeneinträge definieren wie du willst.

Ein Abschluss-Tag `</option>` ist zwar optional, im Hinblick auf verarbeitende Programmiersprachen aber dringend zu empfehlen.

Die Breite der Listenanzeige wird automatisch ermittelt, abhängig vom längsten Eintrag.

Auswahllisten mit Mehrfachauswahl

Wenn du nichts anderes angibst, kann der Anwender aus einer Auswahlliste genau einen Eintrag auswählen. Du kannst eine Mehrfachauswahl erlauben.

Beispiel

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
       "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>W&auml;hlen Sie so viele Favoriten wie Sie wollen!</h1>

<form action="select.htm">
<p>
<select name="top5" size="5" multiple>
<option>Heino</option>
<option>Michael Jackson</option>
<option>Tom Waits</option>
<option>Nina Hagen</option>
<option>Marianne Rosenberg</option>
</select>
</p>
</form>

</body>
</html>
```

Erläuterung

Die Mehrfachauswahl einer Auswahlliste erlaubst du durch das zusätzliche Attribut `multiple` im einleitenden `<select>`-Tag.

Eine Mehrfachauswahl ist für Anwender nicht unmittelbar erkennbar. Deshalb solltest du darauf hinweisen, wenn mehrere Einträge auswählbar sind. Auch ist nicht allen

Anwendern klar, wie sie mehrere Einträge selektieren können. Auf modernen PC-Tastaturen geschieht das normalerweise durch Halten der [Strg]-Taste bei gleichzeitigem Anklicken der gewünschten Listeneinträge.

Wenn du XHTML-konform arbeiten willst, musst du das Attribut in der Form `multiple="multiple"` notieren.

Einträge vorselektieren

Wenn du nichts anderes angibst, ist zunächst kein Eintrag einer Auswahlliste vorselektiert. Du kannst einen Eintrag vorselektieren. In Verbindung mit Mehrfachauswahl kannst du auch mehrere Einträge vorselektieren. Vorselektierte Einträge haben einen sichtbaren Markierungsbalken.

Beispiel

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
       "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Sie k&ouml;nnen auch einen anderen Favoriten w&auml;hlen!</h1>

<form action="select.htm">
<p>
<select name="top5" size="5">
<option>Heino</option>
<option>Michael Jackson</option>
<option selected>Tom Waits</option>
<option>Nina Hagen</option>
<option>Marianne Rosenberg</option>
</select>
</p>
</form>

</body>
</html>
```

Erläuterung

Um einen Eintrag der Auswahlliste vorzuselektieren, gib im einleitenden `<option>`-Tag des betreffenden Eintrags das Attribut `selected` an.

Wenn du XHTML-konform arbeiten willst, musst du das Attribut in der Form `selected="selected"` notieren.

Absendewert von Einträgen bestimmen

Normalerweise wird beim Absenden des Formulars der Text eines ausgewählten Listeneintrags übertragen, der zwischen `<option>` und `</option>` notiert ist. Du kannst jedoch bestimmen, dass intern ein anderer Text übertragen wird.

Beispiel

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
       "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Pizzeria Fantasia</h1>

<form action="select.htm">
<p>Ihre Pizza-Bestellung:</p>
<p>
<select name="Pizza" size="5"
onchange="alert(this.form.Pizza.options[this.form.Pizza.selectedIndex].v
alue)">
<option value="P101">Pizza Napoli</option>
<option value="P102">Pizza Funghi</option>
<option value="P103">Pizza Mare</option>
<option value="P104">Pizza Tonno</option>
<option value="P105">Pizza Mexicana</option>
<option value="P106">Pizza Regina</option>
<option value="P107">Pizza de la Casa</option>
<option value="P108">Pizza Calzone</option>
<option value="P109">Pizza con tutti</option>
</select>
</p>
</form>

</body>
</html>
```

Erläuterung

Um für einen Eintrag der Auswahlliste einen anderen Absendewert zu bestimmen, gib im eineitenden `<option>`-Tag des betreffenden Eintrags das Attribut `value=` an (*value = Wert*). Als Wert weist du den gewünschten Absendewert zu. Beim Absenden des Formulars wird dann der hier bestimmte Text eines ausgewählten Eintrags übertragen, nicht der Text, der dem Anwender beim Listeneintrag angeboten wurde.

Im obigen Beispiel ist im einleitenden `<select>`-Tag ein so genannter Event-Handler notiert. Diesem wird im Beispiel eine JavaScript-Anweisung zugewiesen, die bewirkt, dass ein Meldungsfenster den internen Absendewert des Eintrags zur Kontrolle ausgibt, sobald der Anwender einen Eintrag selektiert.

Event-Handler (*Ereignis-Behandler*) sind ein wichtiges Bindeglied zwischen HTML und JavaScript. Event-Handler werden meist in Form von Attributen in HTML-Tags notiert. Da es sich um Bestandteile handelt, die innerhalb von HTML vorkommen, hat das W3-Konsortium die Event-Handler mittlerweile auch in den HTML-Sprachstandard mit aufgenommen. Dort wird auch festgelegt, in welchen HTML-Tags welcher Event-Handler vorkommen darf. Das Problem dabei ist jedoch, dass die Praxis derzeit noch stark davon abweicht - zumindest bei Netscape 4.x. Der MS Internet Explorer dagegen interpretiert Event-Handler seit seiner Version 4.x weitgehend so universell wie vom W3-Konsortium vorgesehen. Letztendlich hilft aber nur: selber im Einzelfall mit mehreren verschiedenen Browsern testen und ausprobieren.

Event-Handler erkennst du daran, dass solche HTML-Attribute immer mit `on` beginnen, zum Beispiel `onClick=`. Hinter dem Istgleichzeichen notierst du - in Anführungszeichen, eine JavaScript-Anweisung. Wenn du mehrere Anweisungen ausführen willst, dann definierst du dir dazu am besten in einem JavaScript-Bereich eine Funktion

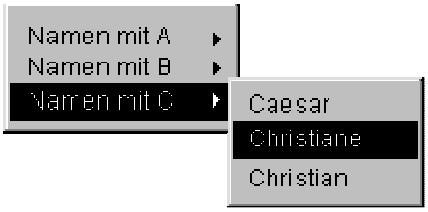
z.B.: `Umrechnen()` und rufst hinter dem Istgleichzeichen diese Funktion auf, also z.B. `onClick="Umrechnen()"`.

Jeder Event-Handler steht für ein bestimmtes Anwenderereignis, `onClick=` etwa für das Ereignis "Anwender hat mit der Maus geklickt". Der Anzeigebereich des HTML-Elements, in dem der Event-Handler notiert ist, ist das auslösende Element. Wenn der Event-Handler `onClick=` beispielsweise in einem Formularbutton notiert wird, wird der damit verknüpfte JavaScript-Code nur dann ausgeführt, wenn der Mausklick im Anzeigebereich dieses Elements erfolgt. Das mag dir jetzt selbstverständlich vorkommen. Ist es auch, solange es beispielsweise um Formularbuttons geht. Aber nach dem erweiterten Modell von HTML 4.0 kann etwa auch ein HTML-Bereich, der mit `<div>...</div>` definiert wird, einen Event-Handler wie `onClick=` enthalten.

Es wurden nur solche Event-Handler aufgenommen, die auch tatsächlich in HTML-Tags vorkommen können und im HTML-4.0-Standard erwähnt sind (mit Ausnahme von `onAbort=` und `onError=`). Das sind weniger, als Netscape und der MS Internet Explorer kennen. Bei Netscape kommt verwirrender Weise hinzu, dass einige Event-Handler zwar so bezeichnet werden, aber eigentlich gar nicht innerhalb von HTML-Tags vorkommen können. Es ist zu hoffen, dass es hierbei in Zukunft mehr Übereinstimmungen zwischen Sprachstandards und Browser-Implementierungen geben wird.

Verschachtelte Auswahllisten (Menüstruktur) definieren

Eine verschachtelte Auswahlliste sollte ein Web-Browser nicht wie üblich als einfache Auswahlliste anzeigen, sondern als verschachtelte Menüstruktur. Anwender moderner grafischer Oberflächen kennen solche verschachtelten Menüs. Das Startmenü von Windows ist beispielsweise ein solches Menü. So in etwa ist es gedacht:



Netscape 6.0 erkennt als erster Browser diese HTML-Elemente, stellt sie aber nicht sehr überzeugend dar.

Beispiel

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
       "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>W&auml;hlen Sie Ihren Lieblingsnamen!</h1>

<form action="select_optgroup.htm">
<p>Zur Auswahl stehen:</p>
<p>
<select name="Namen" size=3>
 <optgroup label="Namen mit A">
  <option label="Anna">Anna</option>
```

```
  <option label="Achim">Achim</option>
  <option label="August">August</option>
 </optgroup>
 <optgroup label="Namen mit B">
  <option label="Berta">Berta</option>
  <option label="Barbara">Barbara</option>
  <option label="Bernhard">Bernhard</option>
 </optgroup>
 <optgroup label="Namen mit C">
  <option label="Caesar">Caesar</option>
  <option label="Christiane">Christiane</option>
  <option label="Christian">Christian</option>
 </optgroup>
</select>
</p>
</form>

</body>
</html>
```

Erläuterung

Für eine verschachtelte Menüstruktur definierst du Untergruppen von Auswahllisten. Dazu notierst du innerhalb von `<select>` und `</select>` für jede gewünschte Untergruppe ein `optgroup`-Element. Zwischen dem einleitenden `<optgroup>`-Tag und dem Abschluss-Tag `</optgroup>` notiere jeweils eine Untergruppe von Einträgen. Im einleitenden `<optgroup>`-Tag vergibst du mit dem Attribut `label=` eine Menübeschriftung für die Gruppe von Einträgen. Dieser Eintrag wird beispielsweise als Menüeintrag mit Pfeiltaste rechts außen angezeigt, um dem Anwender zu signalisieren, dass sich hinter dem Menüeintrag ein Untermenü verbirgt.

Für die Einträge innerhalb einer Untergruppe notiere jeweils ein `option`-Element. Auch hierbei kannst du den anzuzeigenden Menüeintrag mit dem Attribut `label=` bestimmen.

Die Werte, die du hinter `<option>` notierst, sind für die Menüdarstellung ohne Belang. Es ist jedoch wichtig, sie zu notieren, denn Browser, die die Menüdarstellung nicht beherrschen, zeigen stattdessen eine gewöhnliche Auswahlliste mit den Einträgen an, die du hinter den `<option>`-Tags angibst.

Wenn du Unter-Untermenüs realisieren willst, notiere einfach innerhalb von `<optgroup>...</optgroup>` an der gewünschten Stelle einen weiteren Bereich mit `<optgroup>...</optgroup>`.

Radiobuttons

Radiobuttons sind eine Gruppe von beschrifteten Knöpfen, von denen der Anwender einen auswählen kann. Es kann immer nur einer der Radiobuttons ausgewählt sein. Der Wert des ausgewählten Radiobuttons wird beim Absenden des Formulars mit übertragen.

Beispiel

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
       "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>
```

```
<h1>Hier wird abkassiert!</h1>

<form action="input_radio.htm">
<p>Geben Sie Ihre Zahlungsweise an:</p>
<p>
<input type="radio" name="Zahlmethode" value="Mastercard">
Mastercard<br>
<input type="radio" name="Zahlmethode" value="Visa"> Visa<br>
<input type="radio" name="Zahlmethode" value="AmericanExpress"> American
Express
</p>
</form>

</body>
</html>
```

Erläuterung

Radiobuttons werden durch `<input type="radio">` definiert (*input = Eingabe*). Jeder Radiobutton sollte einen internen Bezeichnernamen erhalten, und zwar mit dem Attribut `name=`. Alle Radiobuttons, die den gleichen Namen haben, gehören zu einer Gruppe, d.h. von diesen Buttons kann der Anwender genau einen markieren. Der vergebene Name wird bei CGI-Scripts benötigt, um auf die Daten des Eingabefeldes zugreifen zu können. Bei E-Mail-Formularen taucht der Name als Feldname auf. Und auch bei dem Formularfeldzugriff mit JavaScript ist der Name von Bedeutung.

Mit dem Attribut `value=` bestimmst du einen internen Bezeichnerwert für jeden Radiobutton (*value = Wert*). Wenn der Anwender das Formular abschickt, wird der Bezeichnerwert (value) des markierten Buttons übertragen.

Vor oder hinter dem `<input>`-Tag kannst du den Text notieren, der als Beschriftung der jeweiligen Option erscheint.

Wenn du eine der Auswahlmöglichkeiten vorselektieren willst, dann notiere in dem `<input>`-Tag des entsprechenden Radiobuttons das alleinstehende Attribut `checked`, also z.B.:
`<input type="radio" name="Typ" value="Kassenpatient" checked>`
Wenn du XHTML-konform arbeiten willst, musst du dieses Attribut in der Form `checked="checked"` notieren. Mehr als eine Auswahlmöglichkeit darfst du bei Radiobuttons nicht vorselektieren.

Checkboxen

Checkboxen sind eine Gruppe von ankreuzbaren Quadraten, bei denen der Anwender keine, eins oder mehrere auswählen kann. Die Werte von ausgewählten Checkboxen werden beim Absenden des Formulars mit übertragen.

Beispiel

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
       "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>
```

```
<h1>Pizzabelag nach Wahl!</h1>

<form action="input_checkbox.htm">
<p>Kreuzen Sie die gew&uuml;nschten Zutaten an:</p>
<p>
<input type="checkbox" name="zutat" value="salami"> Salami<br>
<input type="checkbox" name="zutat" value="pilze"> Pilze<br>
<input type="checkbox" name="zutat" value="sardellen"> Sardellen
</p>
</form>

</body>
</html>
```

Erläuterung

Checkboxen werden durch `<input type="checkbox">` definiert (*input = Eingabe*). Jede Checkbox muss einen internen Bezeichnernamen erhalten, und zwar mit der Angabe `name=`. Alle Checkboxen, die den gleichen Namen haben, gehören zu einer Gruppe, d.h. von diesen Elementen kann der Anwender keines, eines oder mehrere ankreuzen.

Mit dem Attribut `value=` bestimmst du einen internen Bezeichnerwert für jede Checkbox (*value = Wert*). Wenn der Anwender das Formular abschickt, werden die Bezeichnerwerte (values) des oder der angekreuzten Buttons übertragen.

Vor oder hinter dem `<input>`-Tag kannst du den Text notieren, der als Beschriftung der jeweiligen Option erscheint.

Wenn du Auswahlmöglichkeiten vorselektieren willst, dann notiere in dem `<input>`-Tag der entsprechenden Checkbox das alleinstehende Attribut `checked`, also z.B.:
`<input type="checkbox" name="Kenntnisse_in" value="HTML" checked>`
Wenn du XHTML-konform arbeiten willst, musst du dieses Attribut in der Form `checked="checked"` notieren.
Bei Checkboxen darfst du mehrere Einträge vorselektieren.

Buttons

Klickbuttons definieren (herkömmlich)

Du kannst Schaltflächen definieren, die keine spezielle Bedeutung haben. Sinnvoll sind solche frei Klickbuttons in Verbindung mit Scriptsprachen wie JavaScript. In der Regel werden sie dazu verwendet, Verweise oder andere JavaScript-gesteuerte Anweisungen auszuführen.

Die hier beschriebene, herkömmliche Methode hat den Vorteil, dass sie auch von älteren Browsern (Netscape seit Version 2.x, MS Internet Explorer seit Version 3.x) interpretiert wird.

Beispiel

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
       "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
```

```
<body>

<h1>Verweise einmal anders</h1>

<form action="input_button.htm">
<p>
<input type="button" name="Verweis" value="SELFHTML Portalseite"
onClick="self.location.href='http://selfaktuell.teamone.de/'">
</p>
</form>

</body>
</html>
```

Erläuterung

Mit `<input type="button">` definierst du einen Button (*input = Eingabe*). Die Beschriftung des Buttons bestimmst du mit dem Attribut `value=` (*value = Wert*). Um anzugeben, was passieren soll, wenn der Button angeklickt wird, kannst du beispielsweise den Event-Handler `onClick=` verwenden. Als Wertzuweisung an das Event-Handler-Attribut kannst du dann JavaScript-Code notieren. Im obigen Beispiel wird mit mit Hilfe der JavaScript-Objekteigenschaft `location.ref` ein Sprungverweis zu dem angegebenen URI ausgeführt. Eine andere beliebte, einfache JavaScript-Anweisung, mit der sich so ein Button belegen lässt, ist `onClick="history.back()"`. Damit erhält der Button die Zurück-Funktion im Browser.

Mit dem Attribut `name=` kannst du einen Namen für den Button vergeben. Unter diesem Namen ist der Button beispielsweise in JavaScript ansprechbar. Der Name sollte nicht zu lang sein und darf keine Leerzeichen, Sonderzeichen oder deutsche Umlaute enthalten. Das erste Zeichen muss ein Buchstabe sein. Danach sind auch Ziffern erlaubt. Benutzen Sie als Sonderzeichen im Namen höchstens den Unterstrich (_), den Bindestrich (-), den Doppelpunkt (:) oder den Punkt (.).

Klickbuttons definieren (modern)

Ab HTML 4.0 dürfen anklickbare Buttons endlich so heißen wie sie heißen: nämlich **Button**. Solche Buttons sind flexibler als herkömmliche Buttons, denn sie dürfen auch einen definierten Inhalt haben, etwa eine Grafik.

Der Nachteil ist, dass diese Sorte Buttons nur von neueren Browsern unterstützt wird - vor allem von Netscape 4.x noch nicht.

Beispiel

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
       "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Verweise einmal anders</h1>

<form action="button.htm">
<div>
<button name="Klickmich" type="button"
```

```
  value="SELFHTML"
onClick="self.location.href='http://selfaktuell.teamone.de/'">
 <p><img src="selfhtml.gif" width="106" height="109" border="0"
alt="SELFHTML Logo"><br>
 <b>SELFHTML Portalseite</b></p>
</button>
</div>
</form>

</body>
</html>
```

Erläuterung

Die Definition eines solchen Buttons leitest du mit `<button>` ein. Dieses Element hat ein End-Tag `</button>`, mit dem du die Definition des Buttons abschließt.

Zwischen dem einleitenden Tag und dem End-Tag können Inhalte stehen. Alles, was du innerhalb von `<button>...</button>` notierst, wird als "Beschriftung" des Buttons angezeigt. Das dürfen durchaus auch eingebundene Grafiken sein, so wie im obigen Beispiel.

Im einleitenden `<button>`-Tag notierst du verschiedene Angaben zum Button. Etwas seltsam erscheint die Angabe `type="button"`, wo doch das Element schon so heißt. Notiere diese Angabe jedoch bei allen Buttons, die du für Scriptsprachen verwendest. Denn mit Hilfe des `button`-Elements kannst du auch zwei andere Button-Typen definieren, nämlich Buttons zum Absenden und Abbrechen.

Mit dem Attribut `name=` kannst du einen Namen für den Button vergeben. Für den Namen gelten die gleichen Bemerkungen wie bei Klickbuttons (herkömmlich).

Mit dem Attribut `value=` kannst du eine Beschriftung für den Button bestimmen, falls du keinen Inhalt innerhalb von `<button>...</button>` notierst. Beachte jedoch, dass die Browser dies ignorieren und bei leerem Inhalt einen mickrigen leeren Button anzeigen.

Um anzugeben, was passieren soll, wenn der Button angeklickt wird, kannst du beispielsweise den Event-Handler `onClick=` verwenden. Als Wertzuweisung an das Event-Handler-Attribut kannst du dann JavaScript-Code notieren.

Da der Button bereits anklickbar ist, sind alle HTML-Elemente darin verboten, die Verweis-Funktion erzeugen. Es sind also keine Verweise beim Inhalt erlaubt, und Grafiken, die als Button-Inhalt angezeigt werden, dürfen kein Attribut `usemap=` für verweis-sensitive Flächen enthalten.

Buttons zum Absenden oder Abbrechen definieren (herkömmlich)

Zwei Standard-Buttons stellt HTML zur Verfügung, um Formulareingaben zu handhaben: einen Button zum "Absenden" und einen zum "Abbrechen". Mit dem Absende-Button kann der Anwender das ausgefüllte Formular losschicken. Mit den Formulardaten geschieht dann das, was im einleitenden `<form>`-Tag bei `action=` festgelegt wurde (siehe hierzu Formular definieren). Mit dem Abbrechen-Button kann der Anwender alle Eingaben verwerfen. Das Formular wird nicht abgeschickt, Eingaben werden gelöscht.

Beispiel

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
       "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Das hier k&ouml;nnen Sie absenden!</h1>

<p>Voraussetzung ist eine Verbindung ins Internet.
Es werden keine Daten gespeichert, das verarbeitende
CGI-Script gibt lediglich die eingelesenen Daten aus.</p>

<form action="http://selfhtml.teamone.de/cgi-bin/comments.pl">
<table border="0" cellpadding="5" cellspacing="0" bgcolor="#E0E0E0">
 <tr>
  <td align="right">Vorname:</td>
  <td><input name="Vorname" type="text" size="30" maxlength="30"></td>
 </tr><tr>
  <td align="right">Zuname:</td>
  <td><input name="Zuname" type="text" size="30" maxlength="40"></td>
 </tr><tr>
  <td align="right" valign="top">Kommentar:</td>
  <td><textarea name="Text" rows="10" cols="50"></textarea></td>
 </tr><tr>
  <td align="right">Formular:</td>
  <td>
   <input type="submit" value=" Absenden ">
   <input type="reset" value=" Abbrechen">
  </td>
 </tr>
</table>
</form>

</body>
</html>
```

Erläuterung

Mit `<input type="submit">` definierst du einen Absende-Button (*input = Eingabe, submit = bestätigen*). Beim Anklicken dieses Buttons werden die Formulardaten abgeschickt, und es wird die Adresse aufgerufen, die im einleitenden `<form>`-Tag beim Attribut `action=` angegeben ist.
Mit `<input type="reset">` definierst du einen Abbrechen-Button (*reset = zurücksetzen*). Eingegebene Daten werden verworfen und nicht abgeschickt.

Mit dem Attribut `value=` bestimmst du die Beschriftung der Buttons (*value = Wert*).

Mit Hilfe von JavaScript kannst du die Eingaben von Anwendern vor dem Absenden des Formulars kontrollieren.

Grafische Buttons zum Absenden definieren

Du kannst innerhalb von Formularen Grafiken referenzieren und diese als Absende-Button definieren. Solche grafische Buttons kannst du als Alternative zu Buttons vom Typ `<input type="submit">` verwenden.

Beispiel

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
       "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Absenden mit Komfort!</h1>

<p>Voraussetzung ist eine Verbindung ins Internet.
Es werden keine Daten gespeichert, das verarbeitende
CGI-Script gibt lediglich die eingelesenen Daten aus.</p>

<form action="http://selfhtml.teamone.de/cgi-bin/comments.pl">
<table border="0" cellpadding="5" cellspacing="0" bgcolor="#E0E0E0">
 <tr>
  <td align="right">Ihre Mailadresse:</td>
  <td><input name="Mail" type="text" size="30" maxlength="30"></td>
 </tr><tr>
  <td align="right">Formular:</td>
  <td>
   <input type="image" src="absende.gif">
  </td>
 </tr>
</table>
</form>

</body>
</html>
```

Erläuterung

Mit `<input type="image">` definierst du einen grafischen Button (*input = Eingabe*). Die gewünschte Grafikdatei bestimmst du mit dem Attribut `src=` (*src = source = Quelle*). Weise eine geeignete Grafik vom Typ GIF, JPEG oder PNG zu. Im obigen Beispiel wird vorausgesetzt, dass sich die Grafik `absende.gif` im gleichen Verzeichnis befindet wie die HTML-Datei mit dem Formular.

Buttons zum Absenden oder Abbrechen definieren (modern)

Dies funktioniert genauso wie das Definieren von Klick-Buttons (modern). Dort wird erklärt, wie solche Buttons definiert werden.

Um einen Button zum Absende-Button (Submit-Button) zu erklären, musst du im einleitenden `<button>`-Tag `type="submit"` notieren. Um den Button zu einem Abbrechen-Button (Reset-Button) zu erklären, musst du `type="reset"` notieren.

Versteckte Elemente in Formularen definieren

Du kannst Felder in einem Formular definieren, die dem Anwender nicht angezeigt werden. Versteckte Felder können Daten enthalten. Beim Absenden des Formulars werden die Daten versteckter Felder mit übertragen. Auf diese Weise kannst du beispielsweise zusätzliche Informationen an CGI-Scripts übergeben oder erläuternden Text einfügen, der bei der E-Mail-Übertragung der Formulardaten in der E-Mail mit enthalten ist.

Auch für JavaScript ist diese Möglichkeit interessant. Da ein JavaScript Formularfelder problemlos auslesen und deren Werte auch ändern kann, ist es auf diese Weise bequem

möglich, interne Daten zu speichern, die nicht am Bildschirm angezeigt werden. So könnte ein JavaScript beispielsweise, nachdem die Seite mit dem Formular beim Anwender geladen ist, Informationen zur Bildschirmauflösung beim Anwender sammeln und die Ergebnisse in ein verstecktes Formularfeld schreiben. Die Daten werden dann, wenn der Anwender das Formular abschickt, mit übertragen.

Beispiel

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
       "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Feedback</h1>

<form name="Feedback" action="http://www.der-allesrausbekommer.de/cgi-
bin/feedback.cgi">
<p>
<input type="hidden" name="UserBrowser" value="">
Ihr Name:<br>
<input name="UserName" type="text" size="40">
Ihr Kommentar:<br>
<textarea name="UserKommentar" rows="10" cols="40"></textarea>
</p>
</form>
<script type="text/javascript">
 document.Feedback.UserBrowser.value = navigator.userAgent;
</script>
</form>

</body>
</html>
```

Erläuterung

Mit `<input type="hidden">` definierst du versteckte Felder in einem Formular (*input = Eingabe, hidden = versteckt*). Die Daten selbst bestimmst du beim Attribut `value=` (*value = Wert*).

Im obigen Beispiel erhält das versteckte Formularfeld zunächst keine Daten (`value=""`). Unterhalb des Formulars ist jedoch ein JavaScript notiert. Dieses Script ermittelt den Browser-Typ, den der Anwender verwendet, und schreibt den ermittelten Wert in das versteckte Formularfeld. Wenn der Anwender das Formular absendet, wird also der verwendete Browser als Formularinhalt mit übertragen.

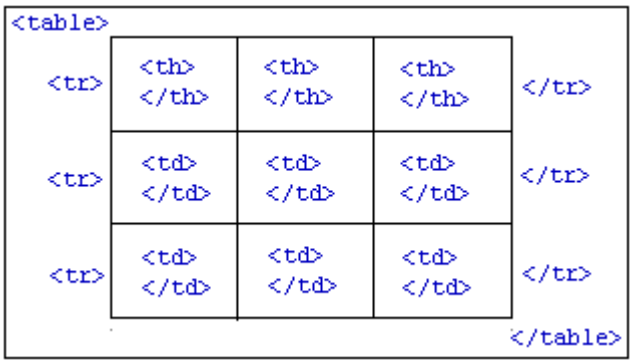
Auf die JavaScript-Anweisung wird hier nicht weiter eingegangen.

Tabellen

Du kannst in HTML Tabellen definieren, um tabellarische Daten darzustellen, oder um Text und Grafik attraktiver am Bildschirm zu verteilen. Obwohl Tabellen natürlich vornehmlich zur Darstellung tabellarischer Daten geschaffen wurden, sind sie in der heutigen Praxis des Web-Designs vor allem als Grundgestaltungsmittel für Seitenlayouts nicht mehr wegzudenken. Rein optisch lässt sich grundsätzlich unterscheiden zwischen Tabellen, die Gitternetzlinien haben (für tabellarische Daten), und Tabellen ohne

sichtbare Gitternetzlinien (so genannte "blinde Tabellen" für mehrspaltigen Text oder für saubere Verteilung von Inhalten auf einer Web-Seite).

Die folgende Grafik zeigt die Wirkung der HTML-Elemente, die eine Tabelle erzeugen:



`<table>` leitet eine Tabelle ein (*table = Tabelle*). Wenn die Tabelle sichtbare Gitternetzlinien enthalten soll, musst du im einleitenden `<table>`-Tag das Attribut `border=` notieren und ihm einen Wert größer `0` zuweisen. Der angegebene Wert ist dann die Breite des Rahmens in Pixeln. Um eine blinde Tabelle ohne sichtbaren Rahmen und Gitternetzlinien zu erzeugen, lasse die Angabe zu `border=` entweder weg, oder - was sauberer ist – du notierst `border="0"`.

`<tr>` leitet eine neue Tabellenzeile ein (*tr = table row = Tabellenzeile*). Im Anschluss daran werden die Zellen (Spalten) der betreffenden Reihe definiert. Am Ende einer Tabellenzeile wird ein abschließendes Tag `</tr>` notiert.

Eine Tabelle kann Kopfzellen und gewöhnliche Datenzellen enthalten. Text in Kopfzellen wird hervorgehoben (meist fett und zentriert ausgerichtet). `<th>` leitet eine Kopfzelle ein, `<td>` eine normale Datenzelle (*th = table header = Tabellenkopf, td = table data = Tabellendaten*). Der Inhalt einer Zelle wird jeweils hinter dem Tag notiert. Obwohl die zugehörigen End-Tags `</th>` bzw. `</td>` offiziell optional sind, ist dringend zu empfehlen, sie immer und in jedem Fall zu notieren.

In einer Tabellenzelle können beliebige Elemente stehen, d.h. außer normalem Text z.B. auch andere Block- und Inline-Elemente. Sogar eine weitere Tabelle kannst du innerhalb einer Zelle definieren.

Die Anzahl der Zellen sollte bei jeder Zeile gleich sein, so dass die Tabelle durchweg die gleiche Anzahl Spalten pro Zeile hat. In der ersten Zeile, die du definierst, legst du deshalb durch die Anzahl der dort definierten Zellen die Anzahl der Spalten Ihrer Tabelle fest.

Tabellenzellen dürfen auch leer sein. Wenn du in einer Zeile für eine Spalte keine Daten eingeben willst, notiere ein einfaches `<td></td>`. Beachte dabei jedoch, dass viele Web-Browser die Zelle in diesem Fall als "nicht vorhanden" darstellt. Probiere deshalb auch mal die Notation `<td> </td>` für leere Tabellenzellen aus.

Spalten vordefinieren

Die Darstellung einer Tabelle ergibt sich zwar automatisch aus den definierten Zeilen und Spalten. Doch für einen Web-Browser ist es nicht ganz einfach, die Darstellung frühzeitig zu ermitteln. Er muss erst die gesamte Tabelle einlesen, bevor er irgendetwas davon darstellen kann. Bei großen Tabellen kann das zu unschönen leeren Bildschirminhalten während des Seitenaufbaus führen.

HTML 4.0 bietet eine Syntax an, um dem Browser gleich zu Beginn der Tabelle mitzuteilen, wie viele Spalten die Tabelle hat, und wie breit diese sind. Dadurch kann der Browser die Tabelle schneller aufbauen, d.h. bereits Teile der Tabelle anzeigen, bevor die gesamte Tabelle eingelesen ist. Ältere Browser ignorieren diese Angaben allerdings.

Beispielschema 1

```
<table border="1">
     <colgroup>
        <col width="80">
        <col width="100">
        <col width="320">
     </colgroup>
    <tr>
      <td>1. Zeile, 1. Spalte</td>
      <td>1. Zeile, 2. Spalte</td>
      <td>1. Zeile, 3. Spalte</td>
    </tr>
        <!-- usw. andere Zeilen der Tabelle -->
</table>
```

Beispielschema 2

```
<table border="1">
     <colgroup width="200" span="3">
     </colgroup>
    <tr>
      <td>1. Zeile, 1. Spalte</td>
      <td>1. Zeile, 2. Spalte</td>
      <td>1. Zeile, 3. Spalte</td>
    </tr>
        <!-- usw. andere Zeilen der Tabelle -->
</table>
```

Beispielschema 3

```
<table border="1" width="100%">
     <colgroup>
        <col width="4*">
        <col width="2*">
        <col width="1*">
     </colgroup>
    <tr>
      <td>1. Zeile, 1. Spalte</td>
      <td>1. Zeile, 2. Spalte</td>
      <td>1. Zeile, 3. Spalte</td>
    </tr>
        <!-- usw. andere Zeilen der Tabelle -->
</table>
```

Erläuterung

Mit `<colgroup>` leitest du hinter dem einleitenden `<table>`-Tag eine Vorab-Definition der Tabellenspalten ein (*colgroup = column group = Spaltengruppe*). Dabei hast du zwei

Möglichkeiten: entweder du möchtest unterschiedlich breite Tabellenspalten haben. Dann gehe so vor wie im obigen Beispielschema 1. Oder du hast eine Tabelle, in der alle Spalten die gleiche einheitliche Breite haben sollen. Dann kannst du so vorgehen wie im obigen Beispielschema 2.

Im Beispielschema 1 enthält das `<colgroup>`-Tag keine weiteren Attribute. Dafür notierst du im Anschluss an `<colgroup>` für jede einzelne gewünschte Tabellenspalte je ein `<col>`-Tag. Das erste `<col>`-Tag definiert die erste Spalte, das zweite die zweite Spalte usw. Wenn du keine weiteren Angaben machst, wird die Breite der Spalten automatisch aufgrund des Tabelleninhalts ermittelt. Mit `width=` [Pixel/Prozent] kannst du jedoch eine Spaltenbreite für die einzelnen Spalten vorgeben (*width = Breite*). Mit `width="100"` erzwingst du beispielsweise eine Spaltenbreite von 100 Pixeln, und mit `width=33%` eine Breite von einem Drittel der Breite der Gesamttabelle.

Im Beispielschema 2 werden keine `<col>`-Tags notiert. Stattdessen notierst du im einleitenden `<colgroup>`-Tag das Attribut `span=` (*span = spannen*). Als Wert weist du die Anzahl der Spalten zu. Mit dem Attribut `width=` kannst du in diesem Fall eine einheitliche Spaltenbreite für alle Spalten definieren.

Bei `width=` hast du neben der Möglichkeit, Pixel oder Prozentwerte anzugeben, auch noch eine dritte Möglichkeit: Du kannst das relative Breitenverhältnis der Spalten untereinander bestimmen, unabhängig davon, wie breit die Tabelle im Verhältnis zum Anzeigefenster ist. Eine solche Möglichkeit stellt das obige Beispielschema 3 vor. Bei Breitenangaben dieser Art weist du `width=` eine Zahl und dahinter ein Sternzeichen zu. Das Sternzeichen ist dabei nur ein Signalzeichen für den Browser, dass er die Zahlen davor nicht als Pixel interpretieren soll. Wichtig sind die Zahlen. Im obigen Beispiel 3 werden drei Spalten definiert, bei denen die relativen Zahlen 4, 2 und 1 in der Summe 7 ergeben. Damit definierst du eine Tabelle, bei der die erste Spalte vier Siebtel der Tabellenbreite einnimmt, die zweite Spalte zwei Siebtel, und die dritte Spalte ein Siebtel. Zur Geltung kommt ein relative Spaltenverhältnis aber erst, wenn du außerdem eine Breite für die gesamte Tabelle angibst. Im obigen Beispielschema 3 geschieht das durch die Angabe `width="100%"` im einleitenden `<table>`-Tag.

`<col>`-Tags dürfen kein Abschluss-Tag haben. Das abschließende `</colgroup>`-Tag ist dagegen optional. Notiere, wenn du XHTML und HTML -konform arbeiten willst, das `<col>`-Tag beispielsweise so: `<col width="4*" />`.

Auch das `<col>`-Tag darf das Attribut `span=` erhalten. Dadurch gruppierst du jedoch nicht mehrere Spalten zu einer, sondern du gibst lediglich an, dass Attribute dieser Spalte auch für die nächsten soundsoviel Spalten gelten sollen. Wenn du beispielsweise `<col span="3" width="100">` notierst, erhalten diese und die nächsten zwei Spalten die Breite von 100 Pixeln.

Das `<colgroup>`-Tag darf die Attribute `span=` und `width=` auch dann enthalten, wenn unterhalb davon `<col>`-Tags definiert werden. Dabei überschreibt die Anzahl der definierten `<col>`-Tags jedoch die Angabe, die mit `<colgroup span=>` gemacht wurde, und das Attribut `width=` innerhalb eines `<col>`-Tags hat Vorrang vor der Angabe `width=` im `<colgroup>`-Tag.

Es ist auch erlaubt, mehrere `<colgroup>`-Tags zu notieren. So kannst du beispielsweise mit `<colgroup width="100" span="3">` und `<colgroup width="50" span="5">`

hintereinander notiert insgesamt 8 Spalten für die Tabelle definieren, wobei die ersten drei Spalten eine Breite von 100 Pixeln erhalten und die nachfolgenden fünf Spalten eine Breite von 50 Pixeln.

Kopf, Körper und Fuß einer Tabelle definieren

Du kannst eine Tabelle logisch in Bereiche aufteilen: einen Kopfbereich, einen oder mehrere Datenbereiche und einen Fußbereich.

Beispiel

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
       "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Spalten vordefinieren</title>
</head>
<body>

<h1>Betroffene Menschen</h1>

<table border="1" rules="groups">
    <thead>
    <tr>
      <th>Assoziation 1</th>
      <th>Assoziation 2</th>
      <th>Assoziation 3</th>
    </tr>
    </thead>
    <tfoot>
    <tr>
      <td><i>betroffen:<br>4 Mio. Menschen</i></td>
      <td><i>betroffen:<br>2 Mio. Menschen</i></td>
      <td><i>betroffen:<br>1 Mio. Menschen</i></td>
    </tr>
    </tfoot>
    <tbody>
    <tr>
      <td>Berlin</td>
      <td>Hamburg</td>
      <td>M&uuml;nchen</td>
    </tr><tr>
      <td>Milj&ouml;h</td>
      <td>Kiez</td>
      <td>Bierdampf</td>
    </tr><tr>
      <td>Buletten</td>
      <td>Frikadellen</td>
      <td>Fleischpflanzerl</td>
    </tr>
    </tbody>
  </table>

</body>
</html>
```

Erläuterung

Den Tabellenkopf leitest du mit `<thead>` ein (*thead = table head = Tabellenkopf*). Daran anschließend kannst du eine oder mehrere Zeilen der Tabelle notieren, die zum Kopfbereich gehören sollen. Mit `</thead>` schließt du den Tabellenkopf ab (das End-Tag ist bei allen Elementen für Tabellenkopf, Tabellenfuß und Tabellenkörper optional, aber dringend zu empfehlen).

Bevor du einen oder mehrere Tabellenkörper-Elemente notierst, musst du hinter dem Tabellenkopf den Tabellenfuß notieren. Diesen leitest du mit `<tfoot>` ein (*tfoot = table foot = Tabellenfuß*). Daran anschließend kannst du eine oder mehrere Zeilen der Tabelle notieren, die zum Fußbereich gehören sollen. Mit `</tfoot>` schließt du den Tabellenfuß ab.

Einen Tabellenkörper leitest du mit `<tbody>` ein (*tbody = table body = Tabellenkörper*). Daran anschließend notierst du den Datenbereich mit einer oder mehreren Tabellenzeilen. Mit `</tbody>` schließt du den Tabellenkörper ab.

Wenn du mit den Elementen `thead`, `tfoot` und `tbody` arbeitest, musst du immer alle drei Elemente verwenden, und zwar immer in der Reihenfolge `thead->tfoot->tbody`. Die Elemente `thead` und `tfoot` dürfen pro Tabelle nur einmal vorkommen, das `tbody`-Element einmal oder beliebig oft.

Label für Elemente

Für Formularelemente wie Eingabefelder oder Auswahllisten gibt es normalerweise keine logische Beschriftungsmöglichkeit. Du kannst zwar Text vor ein solches Element setzen wie "E-Mail-Adresse:", aber ein solcher Text ist normaler HTML-Text, der keinen definierten Bezug zu dem Element hat, für das er als Beschriftung dient. Mit Hilfe von Labels kannst du jedoch einen solchen logischen Bezug zwischen Formularelement und Beschriftungstext herstellen.

Beispiel

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
       "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Meine Beschriftung geh&ouml;rt zu mir wie mein Name an der
T&uuml;r!</h1>

<form action="label.htm">
<table>
  <tr>
    <td><label for="vorname">Vorname:</label></td>
    <td><input type="text" id="vorname" name="Vorname"></td>
  </tr><tr>
    <td><label for="nachname">Zuname:</label></td>
    <td><input type="text" id="nachname" name="Zuname"></td>
  </tr>
</table>
</form>

</body>
</html>
```

Erläuterung

Das Beispiel zeigt, wie du mit Hilfe einer "blinden" Tabelle Beschriftung und Eingabefelder eines Formulars ordentlich formatieren kannst. Das Besondere dabei ist, dass ein logischer Bezug zwischen Beschriftung und Eingabefeld hergestellt wird.

Dazu muss das Formularelement selbst, im obigen Beispiel zwei Eingabefelder, mit Hilfe des Universalattributs `id=` einen eindeutigen Namen erhalten. Vergib keine zu langen Namen. Die Namen dürfen keine Leerzeichen und keine deutschen Umlaute enthalten. Das erste Zeichen muss ein Buchstabe sein. Danach sind auch Ziffern erlaubt. Benutze als Sonderzeichen im Namen höchstens den Unterstrich (_), den Bindestrich (-), den Doppelpunkt (:) oder den Punkt (.).

Den Beschriftungstext, den du einem solchen Formularelement zuordnest, notiere als Inhalt zwischen `<label>` und `</label>`. Der Beschriftungstext kann auch jede Art von HTML/CSS-Formatierung enthalten. Das `label`-Element selbst hat keine sichtbare Wirkung am Bildschirm, es dient lediglich dem Zweck, den logischen Bezug zum Formularelement herzustellen. Diesen Bezug definierst du innerhalb des einleitenden `<label>`-Tags mit dem Attribut `for=` (*for = für*). Dahinter gibst du, in Anführungszeichen, exakt den `id`-Namen des Formularelements an, auf das sich das Label beziehen soll.

Das `label`-Element ist auch auf andere Formularelemente, zum Beispiel auf `textarea` oder `select`, anwendbar.

Tabulator-Reihenfolge

Modernere Browser erlauben es, mit Hilfe der Tabulator-Taste an der Tastatur nacheinander die Elemente eines Formulars anzuspringen.

Normalerweise werden die Formularelemente dabei in der Reihenfolge angesprungen, in der sie in der Datei definiert sind. Du kannst jedoch eine andere Reihenfolge festlegen.

Beispiel

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
       "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Steppen Sie mit [Tab] durch!</h1>

<form action="tabindex.htm">
<pre>
Feld 1: <input type="text" size="40" maxlength="40" tabindex="4">
Feld 2: <input type="text" size="40" maxlength="40" tabindex="1">
Feld 3: <input type="text" size="40" maxlength="40" tabindex="3">
<input type="button" tabindex="2" value="Klick mich">
</pre>
</form>

</body>
</html>
```

Erläuterung

Mit dem Attribut `tabindex=` in einem der Formular-Tags `<input>`, `<textarea>`, `<select>` oder `<button>` kannst du Angaben zur Tabulatorreihenfolge machen. Notiere die Angabe in allen entsprechenden Tags des Formulars und vergib bei jeder Angabe eine

Zahl. Beim Anspringen der Formularelemente mit der Tabulator-Taste wird zuerst das Formularelement mit der niedrigsten Tabindex-Nummer angesprungen, danach dasjenige mit der zweitniedrigsten usw. und als letztes dasjenige mit der höchsten Tabindex-Nummer. Im obigen Beispiel wird also zuerst Feld 2 angesprungen, dann der Button am Ende, dann Feld 3 und zuletzt Feld 1.

Es sind Zahlen zwischen 0 und 32767 erlaubt.

Die Tabindizes beziehen sich stets auf die gesamte angezeigte Datei. Dabei werden auch Verweise, Verweisbereich in verweis-sensitiven Grafiken und Objekte mit einbezogen. Wenn du solche Elemente neben dem Formular auch noch in deiner Datei hast, solltest du die Tabulator-Reihenfolge für alle Elemente gemeinsam festlegen.

Tastaturkürzel

Du kannst dem Anwender anbieten, mit einem Tastendruck zu einem bestimmten Formularelement zu springen. Das Element wird damit angesprungen - "ausgeführt", also zum Beispiel angeklickt oder angekreuzt - wird es dadurch nicht. Ankreuzen von Radiobuttons und Checkboxen funktioniert meist mit der Leertaste, und Anklicken/Ausführen von Buttons mit der Returntaste.

Beispiel

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
       "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Greifen Sie in die Tasten!</h1>

<form action="accesskey.htm">
<p>Ein Feld mit [Alt]+f:<br>
<input type="text" size="40" value="Feld" accesskey="f"></p>
<p>Ein Button mit [Alt]+b:<br>
<input type="button" value="Button" accesskey="b"></p>
<p>Ein Texteingabebereich mit [Alt]+t:<br>
<textarea cols="50" rows="5" accesskey="t">Text</textarea></p>
</form>

</body>
</html>
```

Erläuterung

Mit dem Attribut `accesskey=` kannst du ein Zeichen auf der Tastatur bestimmen, das der Anwender drücken kann, um ein Formularelement direkt anzuspringen (*accesskey = Zugriffstaste*). Bei den meisten Browsern ist es so geregelt, dass du bei `accesskey=` einen Buchstaben angeben kannst, der dann mit `[Alt]` und der Taste für den Buchstaben direkt anwählbar ist.

Erlaubt ist das Attribut `accesskey=` in den Formular-Tags `<input>`, `<textarea>`, `<select>`, `<label>`, `<legend>` oder `<button>`.

Weise den Anwender im Text auf die Möglichkeit des Tastaturzugriffs hin.

Elemente ausgrauen

Du kannst Elemente ausgrauen, um zu signalisieren, dass das Element im aktuellen Zusammenhang ohne Bedeutung ist. Ausgegraute Elemente werden auch bei Tabulator-Sprüngen übergangen.

Sinnvoll ist diese Angabe allerdings vor allem als HTML-Grundlage für Scriptsprachen, die Elemente je nach Eingaben oder Auswahl in anderen Elementen dynamisch ausgrauen. So wäre es beispielsweise sinnvoll, Angaben zum Typ des gefahrenen Autos dynamisch auszugrauen, wenn der Anwender ankreuzt, dass er gar kein Auto besitzt. Als statische Angabe hat das Ausgrauen von Elementen dagegen wohl selten Sinn, denn Formularelemente, die man definiert, definiert man schließlich aus gutem Grund.

Beispiel

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
       "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Ein Formular f&uuml;r sp&auml;ter mal!</h1>

<form action="disabled.htm">
<p>Ihr Name:<br>
<input type="text" size="40" disabled></p>
<p>Ihr Kommentar:<br>
<textarea cols="50" rows="5" disabled></textarea></p>
</form>

</body>
</html>
```

Erläuterung

Mit dem Attribut `disabled` graust du ein Element aus. Erlaubt ist das Attribut in den Formular-Tags `<input>`, `<textarea>`, `<select>`, `<option>`, `<optgroup>` oder `<button>`. Beim Internet Explorer und bei Netscape ist der Effekt, dass das Element einfach nicht editierbar bzw. anklickbar ist. Eine optische Signalisierung gibt es dagegen in beiden Browsern nicht.

Wenn du XHML-konform arbeiten willst, musst du dieses Attribut in der Form `disabled="disabled"` notieren.

Frames

Mit Hilfe von Frames kannst du den Anzeigebereich des Browsers in verschiedene, frei definierbare Segmente aufteilen. Jedes Segment kann eigene Inhalte enthalten. Die einzelnen Anzeigesegmente (also die Frames) können wahlweise einen statischen Inhalt (= "non scrolling regions") oder einen wechselnden Inhalt haben. Verweise in einem Frame können Dateien aufrufen, die dann in einem anderen Frame angezeigt werden.

Frames sind kein weiteres Element, um typische Aufgaben der Textverarbeitung zu bewältigen, sondern ein Element, das die spezifischen Eigenschaften der Bildschirmanzeige konsequent nutzt. Frames eröffnen völlig neue Möglichkeiten, um Information hypertextuell (d.h. nicht-linear) aufzubereiten.

Frames werden ab Netscape 2.0 und ab MS Internet Explorer 3.0 unterstützt. Seit HTML 4.0 gehören die Frames auch zum HTML-Standard. Dort allerdings nicht zur HTML-Variante "Strict", sondern stattdessen mit einer eigenen Variante "Frameset" ausgestattet. Die nebenstehende Abbildung zeigt, wie Frames im Anzeigefenster des Browsers wirken:

Das Bild stellt den Inhalt des gesamten Anzeigefensters des Browsers schematisch dar. Das Anzeigefenster ist in drei unabhängige Bereiche aufgeteilt. Die Verweise links und unten können beispielsweise immer eingeblendet bleiben, während sich der Inhalt des Hauptfensters je nach ausgewähltem Verweis ändern kann. Dabei werden im Beispiel der Abbildung immer drei verschiedene HTML-Dateien gleichzeitig angezeigt: links und unten immer die gleiche Datei, im Hauptfenster jeweils eine wechselnde Datei, je nach ausgewähltem Verweis.

In jedem Frame stehen alle Anzeige-Features zur Verfügung - so ist es z.B. denkbar, in einem Frame Textinformation anzuzeigen, während in einem anderen Frame gleichzeitig ein passendes Video abläuft.

Durch den Einsatz von Frames wachsen die Gestaltungsmöglichkeiten außerordentlich. Frames stellen an das Design von HTML-Seiten aber auch besonders hohe Ansprüche und haben einige nicht zu bestreitende Nachteile.

Grundgerüst einer HTML-Datei mit Framesets

Um Frames zu definieren, brauchst du eine spezielle HTML-Datei, in der ein Frameset definiert wird. Das Frameset bestimmt die Fensteraufteilung. Für eine solche Datei ist ein anderes Grundgerüst erforderlich als das sonst übliche Grundgerüst von HTML-Dateien.

Schema

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
   "http://www.w3.org/TR/html4/frameset.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<frameset ...>  <!-- Frameset-Definition -->
```

```
  <frame ...>   <!-- Framefenster-Definition -->
  <noframes>
    Wird angezeigt, wenn der Browser keine Frames anzeigen kann
  </noframes>
</frameset>
</html>
```

Erläuterung

Die Unterschiede zu anderen HTML-Dateien beginnen bei der Dokumenttyp-Angabe. Bei Dateien, in denen du Framesets und Frames definierst, musst du die oben gezeigte Dokumenttyp-Angabe zur HTML-Variante "Frameset" notieren.

Ein weiterer wichtiger Unterschied zu anderen HTML-Dateien ist, dass Dateien mit Frameset-Definitionen kein `body`-Element besitzen. Anstelle des `body`-Elements, also nach dem abschließenden `</head>`-Tag für den Dateikopf, werden die Frames definiert. Dabei kannst du :

- Framesets definieren
- Frames zu einem Frameset definieren
- Noframes-Bereiche definieren

Mit Framesets bestimmst du die Aufteilung der Framefenster, mit Frames die Datenquellen der einzelnen Framefenster, und der Noframes-Bereich ist für Browser gedacht, die keine Frames anzeigen können, oder bei denen die Anzeige von Frames deaktivierbar ist und vom Anwender deaktiviert wurde.

Der Titel (`<title>...</title>`), den du in der Datei mit der Frameset-Definition angibst, wird während der gesamten Dauer des Frame-Sets angezeigt, auch wenn andere Dateien innerhalb des Frame-Sets aufgerufen werden. Leider gibt es bislang keine Möglichkeit, den Titel zu aktualisieren. Wähle in der Datei, die die Frame-Set-Definitionen enthält, deshalb einen allgemeinen, aussagekräftigen Titel, der für das gesamte Projekt Gültigkeit hat.

Der URI der Datei mit der Frameset-Definition, also beispielsweise http://www.ihr-guter-name.de/, bleibt in der gleichen Form in der Adresszeile des Browsers stehen, auch wenn der Anwender durch Navigieren innerhalb des Framesets andere Seiten des Projekts in eines der Framefenster lädt.

Framesets definieren

Beim Definieren von Framesets bestimmst du, wie das Anzeigefenster des Browsers in verschiedene Framefenster aufgeteilt werden soll. Dabei musst du dir das Anzeigefenster wie den leeren Rahmen einer Tabelle vorstellen. Damit die Tabelle Gestalt annimmt, definierst du Reihen und Spalten für die Framefenster. Ebenso, wie es möglich ist, Tabellen in HTML zu verschachteln, ist das auch bei Framesets möglich. Anstelle eines Frames kannst du auch ein untergeordnetes Frameset notieren. Auf diese Weise kannst du beliebige Fensteraufteilungen erreichen.

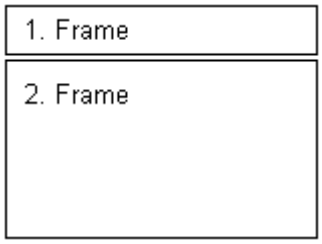
Detailbeispiel 1

```
<frameset rows="20%,80%">
```

```
 <!-- Dadurch ergeben sich zwei Framefenster, deren Inhalt hier bestimmt
wird -->
</frameset>
```

Erläuterung

Die Frameset-Definition im Detailbeispiel 1 ergibt folgende Framefenster-Konstellation:



Im einleitenden Tag `<frameset...>` bestimmst du die Aufteilung. Durch `rows=` teilst du das Anzeigefenster in Reihen auf (*rows = Reihen*). Dahinter bestimmst du, wie die Aufteilung genau aussehen soll. Im Beispiel wird mit Hilfe von `rows="20%,80%"` eine Aufteilung in zwei Reihen erzwungen, wobei die obere Reihe 20% des Anzeigefensters in Anspruch nimmt, die untere 80%.

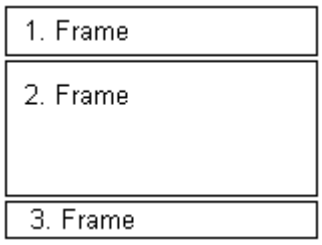
Trenne die Angaben zur Aufteilung durch Kommata!

Detailbeispiel 2

```
<frameset rows="100,*,60">
<!-- Dadurch ergeben sich drei Framefenster, deren Inhalt hier bestimmt
wird -->
</frameset>
```

Erläuterung

Die Frameset-Definition im Detailbeispiel 2 ergibt folgende Framefenster-Konstellation:



Mit der Angabe `rows="100,*,60"` werden drei untereinanderliegende Reihen für Framefenster erzeugt. Die obere Reihe wird genau 100 Pixel hoch, die untere Reihe genau 60 Pixel hoch, und die mittlere Reihe erhält den Rest des Anzeigefensters, abhängig von der Größe des Anzeigefensters beim Anwender.
Mit Zahlenangaben, die ein Prozentzeichen enthalten, wird die Aufteilung also prozentual (relativ zur Größe des Anzeigefensters) interpretiert. Bei Zahlenangaben ohne Prozentzeichen wird die Angabe als absoluter Pixelwert interpretiert. Mit dem Sternzeichen `*` definierst du Bereiche von relativer Größe, abhängig von den übrigen Angaben.

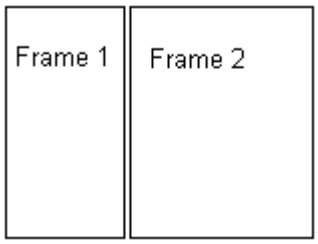
Du kannst auch vier oder mehrere Reihen definieren. So definierst du beispielsweise mit `rows="10%,25%,30%,25%,10%"` fünf Reihen.

Detailbeispiel 3

```
<frameset cols="200,*">
 <!-- Dadurch ergeben sich zwei Framefenster, deren Inhalt hier bestimmt
wird -->
</frameset>
```

Erläuterung

Die Frameset-Definition im Detailbeispiel 3 ergibt folgende Framefenster-Konstellation:



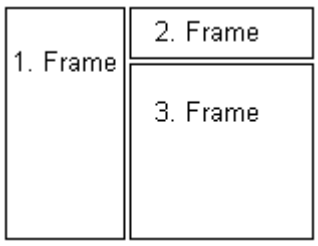
Mit dem Attribut `cols=` teilst du das Anzeigefenster in Spalten auf (*cols = columns = Spalten*). Dahinter bestimmst du, wie die Aufteilung genau aussehen soll. Im Beispiel wird mit Hilfe von `cols="200,*"` eine Aufteilung in zwei Spalten erzwungen, wobei die linke Spalte 200 Pixel des Anzeigefensters in Anspruch nimmt, die rechte den Rest. Bei `cols=` sind die gleichen Angaben möglich wie bei `rows=`.

Detailbeispiel 4

```
<frameset cols="40%,60%">
 <!-- Dadurch ergeben sich zwei Framefenster, deren Inhalt hier bestimmt
wird
         Das erste Framefenster wird normal definiert, anstelle des
zweiten
         folgt jedoch das folgende untergeordnete Frameset: -->
  <frameset rows="20%,80%">
  <!-- Noch mal zwei Framefenster, deren Inhalt hier bestimmt wird -->
  </frameset>
</frameset>
```

Erläuterung

Die Frameset-Definition im Detailbeispiel 4 ergibt folgende Framefenster-Konstellation:



In diesem Beispiel werden Reihen und Spalten für eine komplexere Aufteilung von Framefenstern definiert. Bei solchen Aufteilungen musst du zunächst das gesamte Fenster

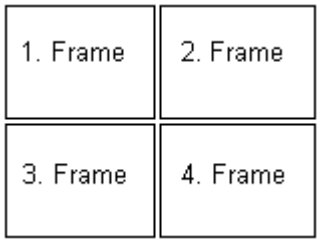
im Auge behalten. Dieses Fenster wird im Beispiel zunächst mit `cols="40%,60%"` in zwei Spalten aufgeteilt. Das erste Framefenster für die linke Spalte (im obigen Beispiel nicht explizit definiert, dort steht nur der längere Kommentar) wird ein ganz normales Framefenster mit Inhalt. Anstelle des zweiten Framefensters für die rechte Spalte wird jedoch ein neues, untergeordnetes Frameset definiert. Mit der Angabe `rows="20%,80%"` erhält es eine Aufteilung in zwei Reihen, also für zwei untereinanderliegende Framefenster, wobei das obere 20% der zur Verfügung stehenden Fläche einnimmt, und das untere 80%.

Detailbeispiel 5

```
<frameset rows="50%,50%">
  <frameset cols="50%,50%">
   <!-- zwei Frames, deren Inhalt hier bestimmt wird -->
  </frameset>
  <frameset cols="50%,50%">
   <!-- noch mal zwei Frames, deren Inhalt hier bestimmt wird -->
  </frameset>
</frameset>
```

Erläuterung

Die Frameset-Definition im Detailbeispiel 5 ergibt folgende Framefenster-Konstellation:



Das Beispiel zeigt eine komplexere Aufteilung in vier Bereiche. Da in einem solchen Fall alle Bereiche gleich groß sind, ist es egal, ob du mit der Definition der Spalten oder Reihen beginnst. Wichtig ist die durchgängige Logik. Wenn du - wie im Beispiel - mit der Definition der Reihen beginnst, musst du anstelle von Frames untergeordnete Frame-Sets definieren, in denen du jede der beiden Reihen in zwei Spalten aufteilst. Erst innerhalb dieser untergeordneten Frame-Sets werden dann die eigentlichen Frames definiert.

Frame-Sets sollten immer so definiert sein, dass das gesamte Anzeigefenster abgedeckt wird. Verwende dazu Prozentangaben, die in der Summe 100 ergeben, oder das Sternzeichen `*`.

Neben der prozentualen Aufteilung ist noch eine weitere relative Angabe zur Framefenster-Aufteilung erlaubt. Du kannst das relative Verhältnis bei `rows=` oder `cols=` auch durch Zahlen, gefolgt von einem Sternzeichen, bestimmen.
Mit einer Aufteilung wie `rows="1*,5*,3*"` definierst du drei Reihen, also Platz für drei untereinanderliegende Framefenster, wobei das oberste Framefenster ein Neuntel einnimmt, das zweite fünf Neuntel und das dritte drei Neuntel (also ein Drittel). Die "9" ergibt sich durch Zusammenzählen von 1, 3 und 5, wodurch das relative Verhältnis ermittelt wird.

Zielfensterbasis

Diese Angabe ist z.B. in Verbindung mit Frames sinnvoll. Du kannst für eine HTML-Datei, die innerhalb eines Framesets in einem Frame-Fenster angezeigt wird, festlegen, dass alle Verweise dieser Datei in einem bestimmten anderen Frame-Fenster angezeigt werden, solange bei einem Verweis kein anderes Frame-Fenster angegeben wird. Da häufig alle Verweisziele einer Datei, die in einem Frame-Fenster angezeigt wird, in einem bestimmten anderen Frame-Fenster angezeigt werden sollen, spart diese einmalige Angabe im Dateikopf viel Tipparbeit und hilft, den Dateiumfang zu verkleinern.

Beispiel

```
<head>
<base target="RechtesFenster">
  <!-- ... andere Angaben im Dateikopf ... -->
</head>
```

Erläuterung

Mit `<base target= ... >` legst du das Default-Fenster fest, in dem Verweisziele angezeigt werden sollen. Voraussetzung ist in dem obigen Beispiel, dass du ein Frameset mit Frame-Fenstern definiert hast. Bei dem gewünschten Frame-Fenster musst du mit dem Attribut `name=` den Fensternamen vergeben haben, den du hier bei `<base target= ... >` angibst.

Anstelle eines selbst vergebenen Frame-Fensternamens kannst du aber auch eine der folgenden festen Angaben notieren, die auch für HTML-Dateien außerhalb von Framesets interessant sind:
`<base target="_blank">` öffnet jeden Verweis der Datei in einem neuen Browser-Fenster.
`<base target="_top">` öffnet jeden Verweis der Datei im gesamten Browserfenster und befreit die Anzeige aus allen eventuell angezeigten Framesets (z.B. auch aus fremden, feindlichen Framesets).
`<base target="_parent">` öffnet jeden Verweis der Datei im übergeordneten Frameset und befreit die Anzeige dem inneren Frameset. `_parent` und `_top` sind dann verschieden, wenn ein Frame-Fenster nochmals ein komplettes Frameset enthält, also eher etwas für Mikro-Freaks.

Frames zu einem Frameset definieren

Nachdem du durch die Definition von Framesets die gewünschte Aufteilung des Anzeigefensters bestimmt hast, gibst du mit der Definition der Frames an, welche Inhalte in den einzelnen Framefenstern zunächst angezeigt werden sollen.

Beispiel Teil 1 - Datei mit Frameset

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
   "http://www.w3.org/TR/html4/frameset.dtd">
<html>
<head>
<title>Text des universellen Titels</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>
```

Beispiel Teil 2 - Datei verweise.htm für linkes Framefenster

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
       "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body bgcolor="#E0C0FF" text="#000000" link="#804080" vlink="#603060"
alink="#804080">

<h1>Navigation</h1>
<p>
<a href="../index.htm" target="Daten"><b>&Uuml;ber Frames</b></a><br>
<a href="../../index.htm" target="Daten"><b>&Uuml;ber HTML</b></a><br>
<a href="../../../index.htm" target="Daten"><b>Noch viel mehr</b></a>
</p>

</body>
</html>
```

Beispiel Teil 3 - Datei startseite.htm für rechtes Framefenster

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
       "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body bgcolor="#FFF0C0" text="#000000" link="#A00000" vlink="#600000"
alink="#A00000">

<h1>Willkommen</h1>
<p>W&auml;hlen Sie links einen Verweis aus!</p>

</body>
</html>
```

Erläuterung

Mit je einem `<frame>`-Tag definierst du innerhalb eines Framesets die zugehörigen Frames (*frame* = *Rahmen*). Theoretisch kannst du das `<frame>`-Tag ohne weitere Attribute notieren. In diesem Fall zeigt der Browser in dem entsprechenden Framefenster eine leere Fläche an. Normalerweise wirst du jedoch die Framefenster mit Inhalt füllen wollen. Dazu dient das Attribut `src=` (*src* = *source* = *Quelle*). Bei der Wertzuweisung an dieses Attribut kannst du die Datei oder Datenquelle angeben, die im entsprechenden Framefenster zunächst angezeigt werden soll. Dabei gelten die Regeln zum Referenzieren in HTML - mit einer Ausnahme: Du darfst keine Anker innerhalb der Datei mit Framesets adressieren. Eine Wertzuweisung wie `src="#weiter_unten"` ist also nicht erlaubt.
Du kannst also andere HTML-Dateien des eigenen Projekts in das Framefenster laden, aber auch andere Dateitypen wie z.B. Grafiken. Auch Dateien oder Datenquellen von entfernten Servern kannst du - mit absoluter `http://`-Adressierung beispielsweise - in

ein Framefenster laden. Doch dabei ist höchste Vorsicht geboten: damit machst du dir fremde Inhalte zueigen und kannst dir schnell juristischen Ärger einhandeln.

Ferner solltest du gleich für jedes definierte Framefenster mit `name=` einen Namen vergeben. Diese Namen brauchst du, um Verweise zu definieren, die in diesem Framefenster angezeigt werden sollen. Ferner sind die Namen wichtig, wenn du zusätzlich JavaScript einsetzen und damit auf einzelne Framefenster zugreifen willst. Der Name sollte nicht zu lang sein und darf keine Leerzeichen, Sonderzeichen oder deutsche Umlaute enthalten. Das erste Zeichen muss ein Buchstabe sein. Danach sind auch Ziffern erlaubt. Benutze als Sonderzeichen im Namen höchstens den Unterstrich (_), den Bindestrich (-), den Doppelpunkt (:) oder den Punkt (.).

Definiere für jede "Zelle" deiner durch das Frameset definierten "Tabelle" ein Framefenster. Im obigen Beispiel wird ein Frameset für zwei Framefenster definiert, mit der Aufteilung, die durch die Angabe `cols="250,*">` bestimmt wird. Ins linke Framefenster, also dem mit 250 Pixeln Breite, soll im Beispiel eine Datei namens `verweise.htm` geladen werden, und ins rechte Framefenster eine Datei namens `startseite.htm`. So wie im Beispiel bei `src=` notiert, müssen sich beide Dateien im gleichen Verzeichnis befinden.

Die beiden HTML-Dateien selbst sind gewöhnliche HTML-Dateien. An der Datei `verweise.htm` kannst du bei den darin notierten Verweisen sehen, wozu die Namen der Framefenster gut sind: mit `target="Daten"` wird dort bei den Verweisen bestimmt, dass das Verweisziel in dem Framefenster angezeigt wird, das bei der Framefenster-Definition mit `name="Daten"` den entsprechenden Namen erhalten hat.

Das `frame`-Element besteht nur aus dem Standalone-Tag `<frame>`. Wenn du **XHTML-konform** arbeitest, musst du das `frame`-Element als inhaltsleer kennzeichnen. Dazu notiere das alleinstehende Tag in der Form `<frame />`.

Noframes-Bereich definieren

Für Anwender, deren Browser keine Frames unterstützen, kannst du dein Projekt so gestalten, dass solche Anwender dennoch Daten aufrufen können. Noframes-Bereiche haben dabei zwei Aufgaben: zum einen können sie in der Datei stehen, in der das Frameset definiert ist. Dort kannst du für Browser, die keine Frames anzeigen können, oder bei denen der Anwender die Anzeige von Frames abgeschaltet hat, einen alternativen Inhalt anbieten. Es kann sich um beliebige, mit HTML strukturierte Inhalte handeln.
Darüber hinaus kannst du Noframes-Bereiche aber auch in HTML-Dateien verwenden, die innerhalb des Framesets als Inhalte in Framefenstern angezeigt werden sollen. Browser, bei denen die Frames angezeigt werden, sollten den Inhalt solcher Noframes-Bereiche unterdrücken. Browser, bei denen die Frames nicht angezeigt werden können oder keine Frames angezeigt werden, zeigen den Inhalt an. Auf diese Weise können Sie z.B. eine einfache Alternativ-Navigation für nicht-framefähige Browser anbieten. Das folgende Beispiel zeigt die Verwendungsweisen für Noframes-Bereiche.

Beispiel Teil 1 - Datei mit Frameset

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

```
<html>
<head>
<title>Text des universellen Titels</title>
</head>
<frameset cols="250,*">
  <frame src="verweise2.htm" name="Navigation">
  <frame src="startseite2.htm" name="Daten">
  <noframes>
    <h1>Willkommen!</h1>
    <p>Dieses Projekt verwendet Frames. Bei Ihnen werden keine Frames
angezeigt.</p>
    <p>W&auml;hlen Sie einen der Verweise aus:<br>
    <a href="startseite2.htm"><b>Startseite</b></a><br>
    <!-- weitere Verweise usw. --></p>
  </noframes>
</frameset>
</html>
```

Beispiel Teil 2 - Datei startseite2.htm für rechtes Framefenster

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
       "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body bgcolor="#FFF0C0" text="#000000" link="#A00000" vlink="#600000"
alink="#A00000">

<h1>Willkommen</h1>
<p>W&auml;hlen Sie links einen Verweis aus!</p>

<noframes>
<p>Dieses Projekt verwendet Frames. Bei Ihnen werden keine Frames
angezeigt:</p>
<p><a href="frames2.htm">zur&uuml;ck zur Startseite</a></p>
</noframes>

</body>
</html>
```

Erläuterung

Das Beispiel zeigt die Datei mit der Frameset-Definition und die Datei, die im rechten Framefenster angezeigt wird. Die Datei `verweise.htm`, die ins linke Framefenster geladen wird, ist in diesem Zusammenhang hier unerheblich und ist deshalb nicht nochmal dargestellt.

Innerhalb von Dateien, die Framesets definieren, sollte ein Noframes-Bereich, markiert durch `<noframes>` und `</noframes>`, am Ende des äußersten Framesets vor dem letzten abschließenden `</frameset>`-Tag notiert werden. Im obigen Beispiel wird nur ein einfaches Frameset definiert, das aus zwei Framefenstern besteht. In diesem Fall werden also die beiden Framefenster notiert und unterhalb davon, vor dem abschließenden `</frameset>`, der Noframes-Bereich.

Innerhalb des Noframes-Bereichs kannst du alles notieren, was du sonst auch zwischen `<body>` und `</body>` notieren kannst, also z.B. Überschriften, Textabsätze, Tabellen, Grafikreferenzen usw. Da es sich bei Browsern, die keine Frames anzeigen können, in der Regel um sehr einfach gestrickte Browser handelt, oder um solche, die unter nicht-grafischen Oberflächen laufen, brauchst du keinen übermäßigen Aufwand für die Optik des Inhalts im Noframes-Bereich zu betreiben. Eine saubere Strukturierung genügt.

Im obigen Beispiel ist innerhalb des Noframes-Bereichs der Datei mit der Frameset-Definition eine Willkommens-Überschrift notiert, ein Hinweis, dass es sich um ein Projekt mit Frames handelt, und es werden Verweise angeboten. Beispielsweise der Verweis auf die Seite `startseite2.htm`, die bei Anzeige von Frames im rechten Framefenster erscheint. Ferner könntest du an dieser Stelle all jene Verweise anbieten, die bei Anzeige von Frames in der links angezeigten Datei `verweise2.htm` stehen. Auf diese Weise können Anwender ohne frame-fähigen Browser dennoch alle Verweise ausführen, die sonst im Frameset angeboten werden.

In der Datei `startseite2.htm` taucht im obigen Beispiel jedoch auch ein Noframes-Bereich auf. Für Browser, die keine Frames anzeigen können, kannst du in solchen Bereichen Inhalte anbieten, die sonst über das Frameset erledigt werden - beispielsweise einen Rückverweis auf die Startseite. Diese Möglichkeit wird auch im Beispiel benutzt: durch den Rückverweis auf die Datei mit den Frameset-Definitionen, die ja auch einen Noframes-Bereich enthält und dort Verweise auf die übrigen Projektseiten anbietet, kannst du eine vollständige, wenn auch etwas simple Alternativ-Navigation für Anwender ohne frame-fähigen Browser realisieren.

Noframes-Bereiche in Projektdateien außerhalb der Datei mit Frameset-Definitionen sind allerdings mit Vorsicht zu genießen: denn leider werden sie nicht nur von nicht-frame-fähigen Browsern angezeigt, sondern auch von älteren frame-fähigen. So zeigt beispielsweise Netscape 4.x den Noframes-Bereich in der Datei `startseite2.htm` des obigen Beispiels gnadenlos an und verwirrt damit Anwender, bei denen das Frameset angezeigt wird. Internet Explorer 5.x und Netscape 6.x unterdrücken dagegen Noframes-Bereiche in solchen Projektdateien.

Wenn du in Projektdateien, die in einem Framefenster angezeigt werden, Noframes-Bereiche notierst, musst du für solche Projektdateien die HTML-Variante "Transitional" wählen, denn in der "Strict"-Variante gibt es keine Frames und auch keine Noframes-Bereiche.

Sinnvolle Einsatzmöglichkeiten für Frames

Frames - Informationsverteilung auf mehrere, voneinander unabhängige Fenster - sind ein faszinierendes Werkzeug, aber auch ein Werkzeug, mit dem man viel verkehrt machen kann. Generell gilt: jeder Einsatz von Frames muss gerechtfertigt sein. Das bedeutet: die Verwendung der Frame-Technik muss dem Anwender als sinnvoll und vorteilhaft erscheinen. Wer Frames aus purem Selbstzweck einsetzt, muss damit rechnen, als technikverliebter HTML-Neuling statt als souveräner Web-Designer betrachtet zu werden.

Der Grund dafür ist, dass Frames nicht wegzudiskutierende Nachteile haben:

- **Probleme bei nicht Frame-fähigen Web-Browsern:**
  Frames werden nicht von allen Web-Browsern angezeigt. Da die gesamte Struktur eines auf Frames basierenden Projekts von der Struktur herkömmlicher Projekte entscheidend abweicht, kann ein Anbieter von Frames solchen Anwendern, die keine Frames anzeigen können, entweder gar keine Alternative anbieten, oder eine "zweigleisige" Alternative, die sehr aufwendig zu realisieren und zu pflegen ist.

- **Frames und Bildschirmauflösung**
  Bei kleineren Bildschirmen, z.B. bei Notebooks oder Handheld-Computern, sind mehr als zwei Framefenster bereits eine Zumutung fürs Auge und die Übersicht.
- **Ladezeiten**
  Frames verlangen mehr HTTP-Kommunikation zwischen Browser und Server, da insgesamt mehr Dateien geladen werden müssen. Im Web kann es unter ungünstigen Verhältnissen leichter zu längeren Ladezeiten kommen.
- **Problematisches Direktansteuern von untergeordneten Seiten**
  Es ist zwar theoretisch möglich, aber meistens nicht im Sinne des Anbieters, wenn andere Anwender ein Lesezeichen oder einen Verweis auf eine HTML-Datei setzen, die Teil eines Framesets ist. Das ist in vielen Fällen ärgerlich. So wird beispielsweise anderen Informationsanbietern die Möglichkeit genommen, in einem bestimmten Informationszusammenhang auf eine bestimmte Seite in einem fremden Projekt zu verweisen.

In folgenden Fällen ist der Einsatz von Frames für den Anwender am ehesten nachvollziehbar:

- **zum schnellen Wechseln zwischen Informationseinheiten**
  In diesem Fall enthält ein Framefenster ein umfangreiches Verzeichnis mit anklickbaren Verweisen auf einzelne Informationsseiten, die in einem anderen, festen Framefenster angezeigt werden. Das "Inhaltsverzeichnis" bleibt also jederzeit eingeblendet, und der Anwender kann zu jedem Zeitpunkt einen neuen Verweis daraus auswählen. Das erspart dem Anwender den wiederholten Rücksprung von den einzelnen Informationsseiten auf das übergeordnete Verzeichnis.
  Beispiel: ein Anbieter von Ferienappartements kann alle zur Verfügung stehenden Objekte in einem Verweis-Verzeichnis auflisten und das jeweils ausgewählte Objekt in einem festen anderen Framefenster anzeigen.
- **zum ständigen Einblenden projektglobaler Steuerverweise**
  Bei umfangreichen Projekten, in denen dem Anwender das Gefühl des "lost in hyperspace" droht, ist es sinnvoll, in einem separaten Framefenster immer gültige Steuerverweise anzubieten, z.B. zur Homepage, zur nächsthöheren logischen Ebene, zum Stichwortverzeichnis oder zur Suchdatenbank. Bei kleinen Projekten, die nur aus einer Handvoll Seiten bestehen, wirkt diese Technik dagegen übertrieben und vermittelt dem Anwender eine falsche Vorstellung von der Größe des Projekts. Wenn der Anwender in einem solchen Fall nach wenigen Mausklicks feststellt, dass er bereits alles gesehen hat, wird er umso enttäuschter sein.
  Beispiel: Eine Zeitung, die im Web ein großes Archiv mit älteren Artikeln anbietet, könnte mit Hilfe der Frame-Technik ständige Verweise zu einem thematisch sortierten Zugangsverzeichnis, zu einem Stichwortverzeichnis und einer Volltext-Suchdatenbank für die einzelnen Artikel anbieten.
- **zum gleichzeitigen Anzeigen von zu vergleichenden Informationen**
  Hypertext bedeutet nicht nur, dem Anwender per Mausklick weitere Informationen zur Verfügung zu stellen, sondern auch, dem Anwender die Möglichkeit zu bieten, sich selbst Informationen so zusammenzustellen, dass er sie optimal miteinander vergleichen und daraus Schlüsse oder Entscheidungen ableiten kann. Zu diesem Zweck eignet sich die Frame-Technik hervorragend, da sie es erlaubt, verschiedene, getrennt voneinander gespeicherte Informationen auf Anwenderwunsch gleichzeitig anzuzeigen.

Beispiel: Eine Verbraucherberatung könnte in einem viergeteilten Frameset in zwei Frames zwei gleichartig aufgebaute Verweis-Verzeichnisse zu Produkttests anbieten. Im dritten Framefenster wird der Produkttest angezeigt, den der Anwender im ersten Framefenster mit Verweisen auswählt; im vierten Framefenster kann der Anwender einen Produkttest anzeigen, den er im zweiten Framefenster mit Verweisen auswählt. Auf diese Weise kann der Anwender beliebige getestete Produkte direkt miteinander vergleichen. Voraussetzung hierzu ist natürlich, dass alle Produkttests einen einheitlichen Aufbau und ein einheitliches Bewertungsschema haben, um direkte Vergleiche zu erlauben.

- **bei besonders kunstvoller Seitengestaltung**
  In diesem Fall muss der Anwender auf den ersten Blick erkennen können, dass die Frame-Technik eine bestimmte künstlerische Aussage unterstützen soll.

Bei Verwendung von Frames besteht auch noch eher als bei "einfachen" Seiten die Gefahr eines gestalterischen Overkills. Achte bei Frames unbedingt darauf, dass die Farben der Inhalte in den verschiedenen Framefenstern insgesamt miteinander harmonieren. Das Gleiche gilt auch für andere optische Eigenschaften wie Schriftarten, Schriftgrößen usw. Ein Navigationsfenster muss nicht krampfhaft "anders" aussehen als ein inhaltstragendes Fenster, damit auch ja jeder Besucher sieht, dass es sich um Frames handelt. Kontraste zwischen Framefenstern dürfen nicht die Regeln einer homogenen Gesamtgestaltung verletzten.

Allgemeines zu Objekten in HTML

Unter "Objekt" wird hier jede Art von Datei oder Datenquelle verstanden, die sich außerhalb einer HTML-Datei befindet und in diese HTML-Datei eingebunden werden soll. Es kann sich um eine Datendatei handeln, also etwa um eine Excel-Tabelle, eine AutoCad-Zeichnung, eine Midi-Musikdatei, ein Flash-Movie, eine Streaming-Quelle für Rundfunkübertragung und vieles andere mehr. Es kann sich aber auch um eine vom Web-Browser ausführbare Datei handeln, also um ein Programm. Das können zum Beispiel Java-Applets oder ActiveX-Controls sein.

Damit nicht für jede neue Form der Einbindung anderer Ressourcen ein neues HTML-Element benötigt wird, soll ein einziges, mächtiges HTML-Element für alle Multimedia- und Fremdprogrammreferenzen genügen: das `object`-Element. Dieses Element kann zwar nicht das Problem lösen, wie eine beliebige Datei beim Anwender angezeigt werden kann, aber es bietet zumindest eine einheitliche Syntax und trägt dadurch zur Vereinfachung von HTML bei.

Das `object`-Element wird jedoch immer noch nicht vollständig von den Web-Browsern unterstützt. Das liegt zum Teil allerdings auch an der Vielzahl der proprietären Multimediaformate und ihren unterschiedlichen Schnittstellen. Das `object`-Element ist jedoch so ausgelegt, dass es möglichst flexibel auf alle denkbaren Anforderungen reagieren kann und dem Browser genügend Information liefert, um mit der ausführenden Anwendung zu kommunizieren und sie in sein Anzeigefenster einzubetten.

Objekte sind aus HTML-Sicht Inline-Elemente. In der "Strict"-Variante von HTML müssen solche Elemente innerhalb von Block-Elementen vorkommen, etwa in einem Textabsatz oder einem allgemeinen Bereich oder auch einer Tabellenzelle.

Das `object`-Element wird vom Internet Explorer seit Version 3.x und von Netscape seit

Version 4.x unterstützt. Von einer ernstzunehmenden Implementierung kann bei diesen Produktversionen aber noch keine Rede sein. Deshalb ist bei den Beispielen in der Regel Internet Explorer 5.0 und Netscape 6.0 angegeben. Aber auch das bedeutet noch lange nicht, dass mit diesen Browsern alles reibungslos funktioniert.

Binde Multimedia-Dateien im Normalfall nicht kommentarlos in deine Web-Seiten ein, sondern weise den Anwender darauf hin, um welche Art von Daten es sich handelt, und unter welchen Voraussetzungen eine korrekte Anzeige möglich ist.

Wenn du große Dateien einbettest, weise den Anwender im umgebenden Text auf die Größe der Datei hin.

Datendateien als Objekt einbinden

Du kannst eine beliebige Datendatei als Objekt in eine HTML-Datei einbinden, also z.B. ein Video, eine Konstruktionszeichnung, eine als fertige Datei vorliegende 3D-Welt, eine Musikdatei oder dergleichen. Auch einfache Textdateien und andere HTML-Dateien kannst du auf diese Weise einbinden. Ein Web-Browser kann solche Dateien anzeigen, wenn er entweder selber in der Lage ist, das Dateiformat anzuzeigen, oder wenn der Anwender ein entsprechendes Plugin installiert hat. Wenn das Plugin installiert ist, kann der Web-Browser die Datei so in seinem Anzeigefenster präsentieren, wie sie von dem Ursprungsprogramm erstellt wurde. Bei Abspielvorgängen, etwa von Videos oder Sound, wird ein entsprechender Player angezeigt - je nachdem, wie das Plugin beschaffen ist.

Wenn dem Browser eine Verknüpfung zwischen dem Datentyp und einem Fremdprogramm bekannt ist, kann er das Fremdprogramm mit der betreffenden Datei starten. Ob die Daten dann jedoch innerhalb des Bereichs auf der Web-Seite angezeigt werden, der für das Objekt definiert ist, hängt davon ab, ob der Browser und das andere Programm entsprechende Kommunikationsmöglichkeiten, z.B. vom Betriebssystem bereitgestellte Kommunikationsschnittstellen, nutzen.

Beispiel

Das Beispiel erfordert mindestens Internet Explorer 5.x oder Netscape 6.x.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
       "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Analoguhr als SVG-Grafik mit eingebettetem JavaScript</h1>

<p><object data="uhr.svg" type="image/svg+xml" width="200" height="200">
Ihr Browser kann das Objekt leider nicht anzeigen!
</object></p>

</body>
</html>
```

Mit `<object>` leitest du die Referenz auf die einzubindende Datei ein, mit `</object>` wird das Element beendet. Das `object`-Element hat keine Pflichtattribute. Je nachdem, was du damit einbindest, musst du selbst herausfinden oder auf Herstellerhinweise

achten, welche der nach HTML-Standard möglichen Attribute du angeben solltest oder musst. Zwischen dem einleitenden `<object>`-Tag und dem abschließenden `</object>`-Tag kann "beliebiger" HTML-Quelltext notiert werden. Im obigen Beispiel ist einfach ein Text notiert, dass der Browser das Objekt nicht anzeigen kann. Ebensogut könnte an dieser Stelle auch eine alternative Grafik notiert werden oder ein alternativer Versuch, das Objekt einzubinden. Es ist sogar erlaubt, als Inhalt eines `object`-Elements ein anderes `object`-Element zu notieren, um damit beispielsweise ein alternatives Objekt in einem anderen Dateiformat einzubinden.

Um eine Datendatei einzubinden, ist das Attribut `data=` vorgesehen. Damit referenzierst du die gewünschte Datei (*data = Daten*). Wenn sich die Datei also beispielsweise im gleichen Verzeichnis wie die HTML-Datei befindet, dann genügt einfach die Angabe des Dateinamens - so wie im obigen Beispiel. Das Referenzieren mit relativen oder absoluten Pfadangaben ist jedoch ebenso möglich, z.B.:

```
<object data="verzeichnis/datei.xy">
<object data="../datei.xy">
<object data="../woanders/datei.xy">
```

Auch das Referenzieren von einem vollständigen URI ist möglich, z.B.:

```
<object data="http://www.tolles-projekt.de/datei.xy">
```

Zusätzlich kannst du den Mime-Type der Datei angeben. Bei Datendateien, die mit `data=` referenziert werden, notiere dazu das Attribut `type=` - so wie im obigen Beispiel. Als Wert weist du einen gültigen Mime-Type zu. Wenn du den Mime-Typ nicht kennst, lasse die Angabe `type=` weg. Wenn du den Mime-Type kennst und angibst, hilfst du dem Web-Browser durch die Angabe, schneller zu reagieren. Eine Übersicht über die gängisten Mime-Typen findest du im Anhang.

Bei eingebundenen Objekten solltest du stets Angaben zu Breite und Höhe notieren. Mit dem Attribut `width=` legst du die Breite in Pixel oder in Prozent in Bezug auf die verfügbare Breite fest, und mit `height=` die Höhe.
Manche Daten, zum Beispiel Videos, haben eine Originalbreite und Originalhöhe. Um so ein Video optimal einzubinden, solltest du dessen genaue Breite und Höhe kennen und mit `width=` und `height=` beim Einbinden der Videodatei angeben.
Manche Plugins, zum Beispiel Sound-Player, haben eine bestimmte Größe. Wenn du die genaue Größe kennst, solltest du beim Einbinden von Sound-Dateien, die mit dem betreffenden Plugin wiedergegeben werden sollen, die genaue Höhe und Breite des Players angeben. So kannst du den Player vollständig und optimal innerhalb der Bildschirmanzeige der HTML-Datei platzieren. Die Größenangaben sind bei solchen Plugins normalerweise bei der Software dokumentiert.
Wenn du zunächst nicht weißt, wie viel Breite oder Höhe das Objekt benötigt, musst du verschiedene Werte ausprobieren.
Mit den Angaben `width="0"` und `height="0"` kannst du die sichtbare Anzeige auch ganz unterdrücken. Das kann zum Beispiel bei Hintergrundmusik erwünscht sein.

Das `object`-Element darf auch im Dateikopf einer HTML-Datei, also zwischen `<head>` und `</head>` notiert werden. Das ist dann sinnvoll, wenn die Datendatei nicht angezeigt werden soll, z.B. wenn einfach eine Hintergrundmusik abgespielt werden soll. In diesem Fall empfiehlt es sich, die Angaben `width="0"` und `height="0"` im einleitenden `<object>`-Tag zu notieren. Bei Objekten, die im HTML-Dateikopf eingebunden werden, darf jedoch zwischen `<object>` und `</object>` kein alternativer Inhalt stehen, der visuelle Ausgaben erzeugt.

Das W3-Konsortium sieht im HTML-Standard auch die Möglichkeit vor, das `object`-Element innerhalb eines Formulars zu notieren, um beispielsweise in visueller Interaktion mit dem Anwender Werte zu ermitteln, die dann beispielsweise in versteckten Formularfeldern gespeichert und zusammen mit den Formulardaten übertragen werden. Vorgaben, wie genau die Kommunikation zwischen Objekt und Formularelementen funktionieren soll, gehören jedoch nicht zur HTML-Spezifikation.

Verweis-sensitive Grafiken als Objekt einbinden

Diese Möglichkeit des `object`-Elements ist vor allem für Grafiken vorgesehen, die **nicht** einem der Standardtypen GIF, JPEG oder PNG entsprechen, die üblicherweise mit dem `<img>`-Tag eingebunden werden. Gedacht ist die Möglichkeit beispielsweise für vektorgrafische CAD-Zeichnungen oder andere Grafikformate, die du mit Hilfe des `object`-Elements einbinden kannst.

Beispiel

Beispiel erfordert installierten Lurawave-Viewer)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
       "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<map name="Verweise">
<area shape="poly"
coords="1,90,129,81,202,275,,273,415,202,417,150,311,1,173"
href="http://www.duesseldorf.de/">
</map>

<h1>Eine Stadt am Rhein</h1>

<p>Klicken Sie, falls Sie die Lurawave-Grafik sehen k&ouml;nnen, auf den
Rhein.
Falls nichts passiert, unterst&uuml;tzt der Browser zwar die Anzeige des
Objekts,
aber nicht das Feature verweis-sensitiver Grafiken f&uuml;r Objekte.</p>

<p><object data="ddorf.lwf" type="image/x-wavelet" width="413"
height="417" usemap="#Verweise">
Ihr Browser kann das Objekt leider nicht anzeigen!
</object></p>

</body>
</html>
```

Erläuterung

Um eine verweis-sensitive Grafik als Objekt einzubinden, notiere im einleitenden `<object>`-Tag das Attribut `usemap=`. Das restliche Verfahren ist dann das gleiche wie bei verweis-sensitiven Grafiken.

Java-Applets als Objekt einbinden

Java-Applets sind ausführbare Programme, deren Bildschirmausgaben ein Web-Browser innerhalb seines Anzeigefensters darstellen kann. Applets können z.B. bewegte Animationen (Tricksequenzen) enthalten, Echtzeitabläufe in bewegten Grafiken darstellen (Stichwort: Börsenticker), oder Interaktionen mit dem Anwender austauschen. So werden Java-Applets etwa häufig bei Online-Banking eingesetzt.

Java-Applets müssen in compilierter Form vorliegen, um bei der Referenzierung in einer HTML-Datei ausgeführt werden zu können. Normalerweise haben compilierte Java-Applets die Dateinamenerweiterung `.class`.

Beispiel

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
       "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Wagen Sie sich durchs Labyrinth!</h1>

<p><object classid="java:zmaze3d.class" codetyte="application/java-vm"
width="600"
height="400"></object></p>

</body>
</html>
```

Erläuterung

Mit dem Attribut `classid=` im einleitenden `<object>`-Tag referenzieren Sie die Implementierung des JavaApplets (*classid = class identifier = Klassenbezeichner*). Erlaubt ist ein spezieller URI: Die Wertzuweisung besteht aus der festen Zeichenfolge `java:` - gefolgt von dem Namen der `.class`-Datei (also der ausführbaren Applet-Datei). Im obigen Beispiel wird auf diese Weise die Datei `zmaze3d.class` in der Form `classid="java:zmaze3d.class"` eingebunden.

Wenn sich die Applet-Datei in einem anderen Verzeichnis befindet als die HTML-Datei, in der sie referenziert wird, oder auf einem anderen Internet-Server, dann musst du den Pfad bzw. die Adresse des Server-Rechners und das Verzeichnis angeben, wo sich die Programmdatei befindet (nur das Verzeichnis, nicht mehr den Namen der Programmdatei). Dazu verwendest du - ebenfalls im einleitenden `<object>`-Tag - das Attribut `codebase=`. Bei der Wertzuweisung an dieses Attribut gelten die Regeln zum Referenzieren in HTML. Die Angabe von `codebase=` kann in einigen Fällen auch erforderlich sein, um Teile eines Java-Applets von einem bestimmten Internet-Server nachzuladen.

Ähnlich wie für Datendateien gibt es auch für ausführbare Programmdateien Mime-Typen. Bei Java-Applets sind die Angaben `codetyte="application/java"` oder `codetyte="application/java-vm"` gebräuchlich.

Zwischen `<object>` und `</object>` kannst du im Zusammenhang mit Java-Applets Aufrufparameter notieren. Viele Java-Applets erwarten einen oder mehrere solche Parameter beim Aufruf.

Beispiel

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
       "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Ein Lauftext mit Java</h1>

<p><object classid="java:zticker.class" codetyte="application/java"
width="600" height="60">
<param name="msg" value="Die Energie des Verstehens">
<param name="href" value="../../../index.htm">
<param name="speed" value="5">
<param name="bgco" value="255,255,255">
<param name="txtco" value="000,000,255">
<param name="hrefco" value="255,255,255">
</object></p>

</body>
</html>
```

Erläuterung

Jeder Parameter wird durch `<param name= value=>` angegeben. Beim Attribut `name=` gibst du den Namen des Parameters an. Bei `value=` gibst du den gewünschten Wert des Parameters an, der dem Applet übergeben werden soll.

Bei Java-Applets, die aus mehreren `class`-Dateien bestehen, musst du die Datei angeben, die den Programmstart enthält. Näheres dazu entimmst du der Dokumentation, die solchen Java-Programmen beigefügt sein sollte. Auch die genauen Anweisungen zum Einbinden eines Java-Applets - z.B. zu erwarteten oder erlaubten Parametern – findest du normalerweise in der Dokumentation zu dem Java-Applet.

ActiveX-Controls als Objekt einbinden

ActiveX-Controls können ähnliche Aufgaben wahrnehmen wie Java-Applets. Jedoch sind sie stärker als Java-Applets in der Windows-Welt und der Welt von Microsoft verankert und werden nur vom Microsoft Internet Explorer interpretiert.

Beispiel

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
       "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Eine MIDI-Datei</h1>

<object width="267" height="175"
        classid="CLSID:05589FA1-C356-11CE-BF01-00AA0055595A">
  <param name="filename" value="ritmo.mid">
</object>

</body>
```

```
</html>
```

Erläuterung

Mit dem Attribut `classid=` referenzieren Sie die Implementierung des gewünschten ActiveX-Controls (*classid = class identifier = Klassenbezeichner*). Die Angabe besteht aus der festen Zeichenfolge `CLSID:` - gefolgt von der Bezeichner-ID. Diese ID musst du kennen. Im obigen Beispiel wird ein recht bekanntes ActiveX-Control referenziert, nämlich dasjenige, das unter Windows zum Abspielen von Multimedia-Dateien zuständig ist. Es bindet den Media-Player von Windows in den Bereich des definierten Objekts ein. Mit `classid="CLSID:05589FA1-C356-11CE-BF01-00AA0055595A"` bindest du also ein ActiveX-Control ein, das es erlaubt, Sound- und Videodateien aller bekannten Formate wie WAV, AU, MID, MP3, AVI, MPEG usw. abzuspielen.

Mit dem Attribut `codebase=` kannst du die Internet-Adresse angeben, von der das ActiveX-Control geladen werden kann, falls der WWW-Browser es auf dem Rechner des Anwenders nicht findet (ActiveX-Controls werden nach dem Laden normalerweise in einem Windows-Unterverzeichnis abgespeichert - Dateien *.ocx - und beim erneuten Aufruf auch wieder von dort geladen.

ActiveX-Controls können genauso wie Java-Applets Parameter übergeben bekommen. Diese werden auch genauso notiert, nämlich mit `<param name= value=>` zwischen `<object>` und `</object>`. Das ActiveX-Control im obigen Beispiel erwartet einen Parameter namens `filename`, der bei `value=` die Angabe des gewünschten Dateinamens der abzuspielenden Datei erwartet.

Flash-Anwendungen als Objekt einbinden

Das Einbinden von Flash (Dateien `*.swf`) funktioniert ähnlich wie das Einbinden von Java-Applets oder ActiveX-Controls.

Beispiel

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
       "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Text des Titels</title>
</head>
<body>

<h1>Das gute alte Wurm-Spiel in Flash!</h1>

<p>Erst auf &quot;Start&quot; klicken. Dann Leertaste dr&uuml;cken. Mit
dem Wurm die
angezeigte Zahl ansteuern und treffen. Au&szlig;enr&auml;nder und
Hindernisse nicht
ber&uuml;hren!</p>

<object classid="CLSID:D27CDB6E-AE6D-11cf-96B8-444553540000"
codebase="http://active.macromedia.com/flash2/cabs/swflash.cab#version=4
,0,0,0"
 width="600" height="400">
 <param name="movie" VALUE="nibbles.swf">
 <param name="quality" value="high">
 <param name="scale" value="exactfit">
 <param name="menu" value="true">
 <param name="bgcolor" value="#000040">
```

```
</object>

</body>
</html>
```

Erläuterung

Mit dem Attribut `classid=` referenzierst du die gewünschte Implementierung (*classid = class identifier = Klassenbezeichner*). Die Angabe besteht aus der festen Zeichenfolge `CLSID:` - gefolgt von der Bezeichner-ID. Für Flash musst du stets `classid="CLSID:D27CDB6E-AE6D-11cf-96B8-444553540000"` notieren.

Mit dem Attribut `codebase=` kannst du die Internet-Adresse angeben, von der das Flash-Plugin geladen werden kann, falls es beim Anwender nicht vorhanden ist. Im obigen Beispiel ist die Adresse für Flash 4 angegeben. Wenn du Flash-Anwendungen einsetzt, die mit Flash 5 erstellt wurden, gib `codebase="http://active.macromedia.com/flash5/cabs/swflash.cab#version=5,0,0,0"` an.

Flash-Anwendungen können genauso wie Java-Applets Parameter übergeben bekommen. Diese werden auch genauso notiert, nämlich mit `<param name= value=>` zwischen `<object>` und `</object>`. Der wichtigste Parameter ist der namens `movie`, bei dem du bei `value` die gewünschte `swf`-Datei angibst.

Anhang

Noch offen:
Inline-Element
Block-Element
CGI
Universalattribut

Benannte Zeichen für HTML-eigene Zeichen

Zeichen	Beschreibung	Name in HTML	Unicode in HTML
"	Anführungszeichen oben	&quot;	&#34;
&	Ampersand-Zeichen, kaufmännisches Und	&amp;	&#38;
<	öffnende spitze Klammer	&lt;	&#60;
>	schließende spitze Klammer	&gt;	&#62;

Benannte Zeichen für den Zeichensatz ISO 8859-1

Die HTML-Namen dieser Zeichen sind seit HTML 3.2 verfügbar.

Zeichen	Beschreibung	Name in HTML	Unicode in HTML
	Erzwungenes Leerzeichen		
¡	umgekehrtes Ausrufezeichen	&iexcl;	&#161;
¢	Cent-Zeichen	&cent;	&#162;
£	Pfund-Zeichen	&pound;	&#163;
¤	Währungs-Zeichen	&curren;	&#164;
¥	Yen-Zeichen	&yen;	&#165;
¦	durchbrochener Strich	&brvbar;	&#166;
§	Paragraph-Zeichen	&sect;	&#167;
¨	Pünktchen oben	&uml;	&#168;
©	Copyright-Zeichen	&copy;	&#169;
ª	Ordinal-Zeichen weiblich	&ordf;	&#170;
«	angewinkelte Anführungszeichen links	&laquo;	&#171;
¬	Verneinungs-Zeichen	&not;	&#172;

	kurzer Trennstrich		
®	Registriermarke-Zeichen	®	®
¯	Überstrich	¯	¯
°	Grad-Zeichen	°	°
±	Plusminus-Zeichen	±	±
²	Hoch-2-Zeichen	²	²
³	Hoch-3-Zeichen	³	³
´	Acute-Zeichen	´	´
µ	Mikro-Zeichen	µ	µ
¶	Absatz-Zeichen	¶	¶
·	Mittelpunkt	·	·
¸	Häkchen unten	¸	¸
¹	Hoch-1-Zeichen	¹	¹
º	Ordinal-Zeichen männlich	º	º
»	angewinkelte Anführungszeichen rechts	»	»
¼	ein Viertel	¼	¼
½	ein Halb	½	½
¾	drei Viertel	¾	¾
¿	umgekehrtes Fragezeichen	¿	¿
À	A mit Accent grave	À	À
Á	A mit Accent acute	Á	Á
Â	A mit Circumflex	Â	Â
Ã	A mit Tilde	Ã	Ã
Ä	A Umlaut	Ä	Ä
Å	A mit Ring	Å	Å
Æ	A mit legiertem E	Æ	Æ
Ç	C mit Häkchen	Ç	Ç
È	E mit Accent grave	È	È

É	E mit Accent acute	É	É
Ê	E mit Circumflex	Ê	Ê
Ë	E Umlaut	Ë	Ë
Ì	I mit Accent grave	Ì	Ì
Í	I mit Accent acute	Í	Í
Î	I mit Circumflex	Î	Î
Ï	I Umlaut	Ï	Ï
Ð	Eth (isländisch)	Ð	Ð
Ñ	N mit Tilde	Ñ	Ñ
Ò	O mit Accent grave	Ò	Ò
Ó	O mit Accent acute	Ó	Ó
Ô	O mit Circumflex	Ô	Ô
Õ	O mit Tilde	Õ	Õ
Ö	O Umlaut	Ö	Ö
×	Mal-Zeichen	×	×
Ø	O mit Schrägstrich	Ø	Ø
Ù	U mit Accent grave	Ù	Ù
Ú	U mit Accent acute	Ú	Ú
Û	U mit Circumflex	Û	Û
Ü	U Umlaut	Ü	Ü
Ý	Y mit Accent acute	Ý	Ý
Þ	THORN (isländisch)	Þ	Þ
ß	scharfes S	ß	ß
à	a mit Accent grave	à	à
á	a mit Accent acute	á	á
â	a mit Circumflex	â	â
ã	a mit Tilde	ã	ã
ä	a Umlaut	ä	ä

å	a mit Ring	å	å
æ	a mit legiertem e	æ	æ
ç	c mit Häkchen	ç	ç
è	e mit Accent grave	è	è
é	e mit Accent acute	é	é
ê	e mit Circumflex	ê	ê
ë	e Umlaut	ë	ë
ì	i mit Accent grave	ì	ì
í	i mit Accent acute	í	í
î	i mit Circumflex	î	î
ï	i Umlaut	ï	ï
ð	eth (isländisch)	ð	ð
ñ	n mit Tilde	ñ	ñ
ò	o mit Accent grave	ò	ò
ó	o mit Accent acute	ó	ó
ô	o mit Circumflex	ô	ô
õ	o mit Tilde	õ	õ
ö	o Umlaut	ö	ö
÷	Divisions-Zeichen	÷	÷
ø	o mit Schrägstrich	ø	ø
ù	u mit Accent grave	ù	ù
ú	u mit Accent acute	ú	ú
û	u mit Circumflex	û	û
ü	u Umlaut	ü	ü
ý	y mit Accent acute	ý	ý
þ	thorn (isländisch)	þ	þ
ÿ	y Umlaut	ÿ	ÿ

Benannte Zeichen für griechische Buchstaben

Die HTML-Namen dieser Zeichen sind seit HTML 4.0 verfügbar.

Zeichen	Beschreibung	Name in HTML	Unicode in HTML
Α	Alpha groß	Α	Α
α	alpha klein	α	α
Β	Beta groß	Β	Β
β	beta klein	β	β
Γ	Gamma groß	Γ	Γ
γ	gamma klein	γ	γ
Δ	Delta groß	Δ	Δ
δ	delta klein	δ	δ
Ε	Epsilon groß	Ε	Ε
ε	epsilon klein	ε	ε
Ζ	Zeta groß	Ζ	Ζ
ζ	zeta klein	ζ	ζ
Η	Eta groß	Η	Η
η	eta klein	η	η
Θ	Theta groß	Θ	Θ
θ	theta klein	θ	θ
Ι	Iota groß	Ι	Ι
ι	iota klein	ι	ι
Κ	Kappa groß	Κ	Κ
κ	kappa klein	κ	κ
Λ	Lambda groß	Λ	Λ
λ	lambda klein	λ	λ
Μ	Mu groß	Μ	Μ
μ	mu klein	μ	μ
Ν	Nu groß	Ν	Ν
ν	nu klein	ν	ν
Ξ	Xi groß	Ξ	Ξ
ξ	xi klein	ξ	ξ
Ο	Omicron groß	Ο	Ο
ο	omicron klein	ο	ο
Π	Pi groß	Π	Π
π	pi klein	π	π
Ρ	Rho groß	Ρ	Ρ
ρ	rho klein	ρ	ρ

Σ	Sigma groß	Σ	Σ
ς	sigmaf klein	ς	ς
σ	sigma klein	σ	σ
Τ	Tau groß	Τ	Τ
τ	tau klein	τ	τ
Υ	Upsilon groß	Υ	Υ
υ	upsilon klein	υ	υ
Φ	Phi groß	Φ	Φ
φ	phi klein	φ	φ
Χ	Chi groß	Χ	Χ
χ	chi klein	χ	χ
Ψ	Psi groß	Ψ	Ψ
ψ	psi klein	ψ	ψ
Ω	Omega groß	Ω	Ω
ω	omega klein	ω	ω
ϑ	theta Symbol	ϑ	ϑ
ϒ	upsilon mit Haken	ϒ	ϒ
ϖ	pi Symbol	ϖ	ϖ

Benannte Zeichen für mathematische Symbole

Die HTML-Namen dieser Zeichen sind seit HTML 4.0 verfügbar.

Zeichen	Beschreibung	Name in HTML	Unicode in HTML
∀	für alle	∀	∀
∂	teilweise	∂	∂
∃	existiert	∃	∃
∅	leer	∅	∅
∇	nabla	∇	∇
∈	Element von	∈	∈
∉	kein Element von	∉	∉
∋	enthält als Element	∋	∋
∏	Produkt	∏	∏
∑	Summe	∑	∑
−	minus	−	−
∗	Asterisk	∗	∗
√	Quadratwurzel	√	√

∝	proportional zu	∝	∝
∞	unendlich	∞	∞
∠	Winkel	∠	∠
∧	und	∧	⊥
∨	oder	∨	⊦
∩	Schnittpunkt	∩	∩
∪	Einheit	∪	∪
∫	Integral	∫	∫
∴	deshalb	∴	∴
∼	ähnlich wie	∼	∼
≅	annähernd gleich	≅	≅
≈	beinahe gleich	≈	≈
≠	ungleich	≠	≠
≡	identisch mit	≡	≡
≤	kleiner gleich	≤	≤
≥	größer gleich	≥	≥
⊂	Untermenge von	⊂	⊂
⊃	Obermenge von	⊃	⊃
⊄	keine Untermenge von	⊄	⊄
⊆	Untermenge von oder gleich mit	⊆	⊆
⊇	Obermenge von oder gleich mit	⊇	⊇
⊕	Direktsumme	⊕	⊕
⊗	Vektorprodukt	⊗	⊗
⊥	senkrecht zu	⊥	⊥
⋅	Punkt-Operator	⋅	⋅
◊	Raute	◊	◊

Benannte Zeichen für technische Symbole

Die HTML-Namen dieser Zeichen sind seit HTML 4.0 verfügbar.

Zeichen	Beschreibung	Name in HTML	Unicode in HTML
⌈	links oben	⌈	⌈
⌉	rechts oben	⌉	⌉
⌊	links unten	⌊	⌊
⌋	rechts unten	⌋	⌋
⟨	spitze Klammer links	⟨	〈

⟩	spitze Klammer rechts	⟩	〉

Benannte Zeichen für Pfeil-Symbole

Die HTML-Namen dieser Zeichen sind seit HTML 4.0 verfügbar.

Zeichen	Beschreibung	Name in HTML	Unicode in HTML
←	Pfeil links	←	←
↑	Pfeil oben	↑	↑
→	Pfeil rechts	→	→
↓	Pfeil unten	↓	↓
↔	Pfeil links/rechts	↔	↔
↵	Pfeil unten-Knick-links	↵	↵
⇐	Doppelpfeil links	⇐	⇐
⇑	Doppelpfeil oben	⇑	⇑
⇒	Doppelpfeil rechts	⇒	⇒
⇓	Doppelpfeil unten	⇓	⇓
⇔	Doppelpfeil links/rechts	⇔	⇔

Benannte Zeichen für diverse Symbole

Die HTML-Namen dieser Zeichen sind seit HTML 4.0 verfügbar.

Zeichen	Beschreibung	Name in HTML	Unicode in HTML
•	Bullet-Zeichen	•	•
…	Horizontale Ellipse	…	…
′	Minutenzeichen	′	′
‾	Überstrich	‾	‾
⁄	Bruchstrich	⁄	⁄
℘	Weierstrass p	℘	℘
ℑ	Zeichen für "imaginär"	ℑ	ℑ
ℜ	Zeichen für "real"	ℜ	ℜ
™	Trademark-Zeichen	™	™
€	Euro-Zeichen	€	€
ℵ	Alef-Symbol	ℵ	ℵ
♠	Pik-Zeichen	♠	♠

♣	Kreuz-Zeichen	♣	♣
♥	Herz-Zeichen	♥	♥
♦	Karo-Zeichen	♦	♦

Benannte Zeichen für Interpunktion

Die HTML-Namen dieser Zeichen sind seit HTML 4.0 verfügbar.

Zeichen	Beschreibung	Name in HTML	Unicode in HTML
	Leerzeichen Breite n		
	Leerzeichen Breite m		
	Schmales Leerzeichen		
	null breiter Nichtverbinder	‌	‌
	null breiter Verbinder	‍	‍
	links-nach-rechts-Zeichen	‎	‎
	rechts-nach-links-Zeichen	‏	‏
–	Gedankenstrich Breite n	–	–
—	Gedankenstrich Breite m	—	—
‘	einfaches Anführungszeichen links	‘	‘
’	einfaches Anführungszeichen rechts	’	’
‚	einfaches low-9-Zeichen	‚	‚
“	doppeltes Anführungszeichen links	“	“
”	doppeltes Anführungszeichen rechts	”	”
„	doppeltes low-9-Zeichen rechts	„	„
†	Kreuz	†	†
‡	Doppelkreuz	‡	‡
‰	zu tausend	‰	‰
‹	angewinkeltes einzelnes Anf.zeichen links	‹	‹
›	angewinkeltes einzelnes Anf.zeichen rechts	›	›

Übersicht von Mime-Typen

Mime-Typ	Dateiendung(en)	Bedeutung
application/acad	*.dwg	AutoCAD-Dateien (nach NCSA)
application/applefile		AppleFile-Dateien
application/astound	*.asd *.asn	Astound-Dateien
application/dsptype	*.tsp	TSP-Dateien
application/dxf	*.dxf	AutoCAD-Dateien (nach CERN)
application/futuresplash	*.spl	Flash Futuresplash-Dateien
application/gzip	*.gz	GNU Zip-Dateien
application/listenup	*.ptlk	Listenup-Dateien
application/mac-binhex40	*.hqx	Macintosh Binär-Dateien
application/mbedlet	*.mbd	Mbedlet-Dateien
application/mif	*.mif	FrameMaker Interchange Format Dateien
application/msexcel	*.xls *.xla	Microsoft Excel Dateien
application/mshelp	*.hlp *.chm	Microsoft Windows Hilfe Dateien
application/mspowerpoint	*.ppt *.ppz *.pps *.pot	Microsoft Powerpoint Dateien
application/msword	*.doc *.dot	Microsoft Word Dateien
application/octet-stream	*.bin *.exe *.com *.dll *.class	Ausführbare Dateien
application/oda	*.oda	Oda-Dateien
application/pdf	*.pdf	Adobe PDF-Dateien
application/postscript	*.ai *.eps *.ps	Adobe Postscript-Dateien
application/rtc	*.rtc	RTC-Dateien
application/rtf	*.rtf	Microsoft RTF-Dateien
application/studiom	*.smp	Studiom-Dateien
application/toolbook	*.tbk	Toolbook-Dateien
application/vocaltec-media-desc	*.vmd	Vocaltec Mediadesc-Dateien
application/vocaltec-media-file	*.vmf	Vocaltec Media-Dateien
application/x-bcpio	*.bcpio	BCPIO-Dateien
application/x-compress	*.z	-Dateien
application/x-cpio	*.cpio	CPIO-Dateien
application/x-csh	*.csh	C-Shellscript-Dateien
application/x-director	*.dcr *.dir *.dxr	-Dateien
application/x-dvi	*.dvi	DVI-Dateien
application/x-envoy	*.evy	Envoy-Dateien
application/x-gtar	*.gtar	GNU tar-Archiv-Dateien
application/x-hdf	*.hdf	HDF-Dateien

application/x-httpd-php	*.php *.phtml	PHP-Dateien
application/x-javascript	*.js	serverseitige JavaScript-Dateien
application/x-latex	*.latex	Latex-Quelldateien
application/x-macbinary	*.bin	Macintosh Binärdateien
application/x-mif	*.mif	FrameMaker Interchange Format Dateien
application/x-netcdf	*.nc *.cdf	Unidata CDF-Dateien
application/x-nschat	*.nsc	NS Chat-Dateien
application/x-sh	*.sh	Bourne Shellscript-Dateien
application/x-shar	*.shar	Shell-Archiv-Dateien
application/x-shockwave-flash	*.swf *.cab	Flash Shockwave-Dateien
application/x-sprite	*.spr *.sprite	Sprite-Dateien
application/x-stuffit	*.sit	Stuffit-Dateien
application/x-supercard	*.sca	Supercard-Dateien
application/x-sv4cpio	*.sv4cpio	CPIO-Dateien
application/x-sv4crc	*.sv4crc	CPIO-Dateien mit CRC
application/x-tar	*.tar	tar-Archivdateien
application/x-tcl	*.tcl	TCL Scriptdateien
application/x-tex	*.tex	TEX-Dateien
application/x-texinfo	*.texinfo *.texi	TEXinfo-Dateien
application/x-troff	*.t *.tr *.roff	TROFF-Dateien (Unix)
application/x-troff-man	*.man *.troff	TROFF-Dateien mit MAN-Makros (Unix)
application/x-troff-me	*.me *.troff	TROFF-Dateien mit ME-Makros (Unix)
application/x-troff-ms	*.me *.troff	TROFF-Dateien mit MS-Makros (Unix)
application/x-ustar	*.ustar	tar-Archivdateien (Posix)
application/x-wais-source	*.src	WAIS Quelldateien
application/x-www-form-urlencoded		HTML-Formulardaten an CGI
application/zip	*.zip	ZIP-Archivdateien
audio/basic	*.au *.snd	Sound-Dateien
audio/echospeech	*.es	Echospeed-Dateien
audio/tsplayer	*.tsi	TS-Player-Dateien
audio/voxware	*.vox	Vox-Dateien
audio/x-aiff	*.aif *.aiff *.aifc	AIFF-Sound-Dateien
audio/x-dspeeh	*.dus *.cht	Sprachdateien
audio/x-midi	*.mid *.midi	MIDI-Dateien
audio/x-mpeg	*.mp2	MPEG-Dateien
audio/x-pn-realaudio	*.ram *.ra	RealAudio-Dateien
audio/x-pn-realaudio-plugin	*.rpm	RealAudio-Plugin-Dateien
audio/x-qt-stream	*.stream	-Dateien

audio/x-wav	*.wav	Wav-Dateien
drawing/x-dwf	*.dwf	Drawing-Dateien
image/cis-cod	*.cod	CIS-Cod-Dateien
image/cmu-raster	*.ras	CMU-Raster-Dateien
image/fif	*.fif	FIF-Dateien
image/gif	*.gif	GIF-Dateien
image/ief	*.ief	IEF-Dateien
image/jpeg	*.jpeg *.jpg *.jpe	JPEG-Dateien
image/tiff	*.tiff *.tif	TIFF-Dateien
image/vasa	*.mcf	Vasa-Dateien
image/vnd.wap.wbmp	*.wbmp	Bitmap-Dateien (WAP)
image/x-freehand	*.fh4 *.fh5 *.fhc	Freehand-Dateien
image/x-portable-anymap	*.pnm	PBM Anymap Dateien
image/x-portable-bitmap	*.pbm	PBM Bitmap Dateien
image/x-portable-graymap	*.pgm	PBM Graymap Dateien
image/x-portable-pixmap	*.ppm	PBM Pixmap Dateien
image/x-rgb	*.rgb	RGB-Dateien
image/x-windowdump	*.xwd	X-Windows Dump
image/x-xbitmap	*.xbm	XBM-Dateien
image/x-xpixmap	*.xpm	XPM-Dateien
message/external-body		Nachricht mit externem Inhalt
message/http		HTTP-Headernachricht
message/news		Newsgroup-Nachricht
message/partial		Nachricht mit Teilinhalt
message/rfc822		Nachricht nach RFC 1822
model/vrml	*.wrl	Visualisierung virtueller Welten
multipart/alternative		mehrteilige Daten gemischt
multipart/byteranges		mehrteilige Daten mit Byte-Angaben
multipart/digest		mehrteilige Daten / Auswahl
multipart/encrypted		mehrteilige Daten verschlüsselt
multipart/form-data		mehrteilige Daten aus HTML-Formular (z.B. File-Upload)
multipart/mixed		mehrteilige Daten gemischt
multipart/parallel		mehrteilige Daten parallel
multipart/related		mehrteilige Daten / verbunden
multipart/report		mehrteilige Daten / Bericht

multipart/signed		mehrteilige Daten / bezeichnet
multipart/voice-message		mehrteilige Daten / Sprachnachricht
text/comma-separated-values	*.csv	komma-separierte Datendateien
text/css	*.css	CSS Stylesheet-Dateien
text/html	*.htm *.html *.shtml	-Dateien
text/javascript	*.js	JavaScript-Dateien
text/plain	*.txt	reine Textdateien
text/richtext	*.rtx	Richtext-Dateien
text/rtf	*.rtf	Microsoft RTF-Dateien
text/tab-separated-values	*.tsv	tabulator-separierte Datendateien
text/vnd.wap.wml	*.wml	WML-Dateien (WAP)
application/vnd.wap.wmlc	*.wmlc	WMLC-Dateien (WAP)
text/vnd.wap.wmlscript	*.wmls	WML-Scriptdateien (WAP)
application/vnd.wap.wmlscriptc	*.wmlsc	WML-Script-C-dateien (WAP)
text/xml-external-parsed-entity		extern geparste XML-Dateien
text/x-setext	*.etx	SeText-Dateien
text/x-sgml	*.sgm *.sgml	SGML-Dateien
text/x-speech	*.talk *.spc	Speech-Dateien
video/mpeg	*.mpeg *.mpg *.mpe	MPEG-Dateien
video/quicktime	*.qt *.mov	Quicktime-Dateien
video/vnd.vivo	*viv *.vivo	Vivo-Dateien
video/x-msvideo	*.avi	Microsoft AVI-Dateien
video/x-sgi-movie	*.movie	Movie-Dateien
workbook/formulaone	*.vts *.vtts	FormulaOne-Dateien
x-world/x-3dmf	*.3dmf *.3dm *.qd3d *.qd3	3DMF-Dateien
x-world/x-vrml	*.wrl	VRML-Dateien

Literaturverzeichnis

HTML-Einführung, Hubert Partl, http://www.boku.ac.at/htmleinf Stand 21.12.2004

Übrige Inhalte und Anhang: www.selfhtml.de Stand Dezember 2004