Allgemeines zu XHTML

Während HTML eine relativ neue Erfindung ist, existiert das zugrunde liegende Konzept der Auszeichnungssprachen schon viel länger. HTML ist lediglich eine Auszeichnungssprache unter vielen, allerdings die bislang weltweit erfolgreichste und bekannteste. Ein Merkmal des Konzepts solcher Auszeichnungssprachen ist, dass ihre Bestandteile nicht einfach "frei erfunden" sind, sondern dass es Meta-Sprachen gibt, mit deren Hilfe die Regeln und Bestandteile solcher Auszeichnungssprachen definiert werden.

Bereits seit 1986 gibt es die als ISO 8879 standardisierte Meta-Sprache **SGML** (*Standard Generalized Markup Language = standardisierte verallgemeinerte Auszeichnungssprache*). Diese Meta-Sprache erlaubt das Definieren von Auszeichnungssprachen mit Hilfe so genannter **DTDs** (*Document Type Definitions = Dokumenttyp-Definitionen*). In den DTDs wird festgelegt, welche Elemente eine Auszeichnungssprache hat, welche zugehörigen Attribute, sowie Regeln, welche Elemente innerhalb welcher anderen Elemente vorkommen können usw. Nun ist SGML sehr ausgereift, gilt aber gemeinhin als ziemlich "sophisticated". So entschloss man sich dazu, eine reduziertere Variante von SGML zu etablieren, und zwar unter dem Namen **XML** (*Extensible Markup Language = erweiterbare Auszeichnungssprache*). XML erlaubt ebenso wie SGML das Definieren von Auszeichnungssprachen mit Hilfe von DTDs.

HTML, das Anfang der 90er-Jahre entstand, wurde mit Hilfe von SGML definiert. Bis einschließlich HTML 4.x ist das auch heute noch der Fall. Im Zuge von XML und seiner wachsenden Bedeutung für immer mehr Dateiformate, die auch im Web ihren Einsatz finden, entstand der Wunsch, auch HTML mit Hilfe von XML zu definieren anstatt wie bisher mit SGML. Damit kein Versionenwirrwarr entsteht, entschloss man sich dazu, dieses neue, XML-basierte HTML mit einem neuen Namen und eigener Versionenkontrolle auszustatten. Heraus kam dabei **XHTML** (*Extensible HyperText Markup Language = erweiterbare Hypertext-Auszeichnungssprache*).

Seit Januar 2000 liegt XHTML 1.0 als Empfehlung des W3-Konsortiums vor und hat damit den gleichen verbindlichen Stellenwert wie etwa HTML 4.0. XHTML 1.0 ist nichts anderes als der Versuch, das SGML-basierte HTML 4.0 mit Hilfe von XML "nachzubauen". Dahinter steckt keine Spielerei von Designern künstlicher Sprachen, sondern es gibt gute Gründe dafür:

- **XHTML ist syntaktisch hundertprozentig kompatibel zu wichtigen XML-Standardsprachen wie SVG, WML, SMIL usw.**
- **XHTML kann dadurch als Dateninsel innerhalb anderer Sprachen eingebunden werden, und ebenso ist es möglich, diese anderen Sprachen wieder als Dateninsel innerhalb von XHTML einzubinden. So ist es beispielsweise problemlos möglich, eine SVG-Grafik als Dateninsel in eine XHTML-Datei einzubetten. Die SVG-Grafik muss dadurch nicht mehr referenziert werden, sondern ist Bestandteil der XHTML-Datei.**
- **Durch die gemeinsame syntaktische Grundlage auf der Basis von XML ist es auch möglich, das Auslesen und Verarbeiten über Programmiersprachen zu vereinheitlichen. Genau das geschieht beim DOM (*Document Object Model = Dokument-Objektmodell*). Das DOM definiert Objekte, Eigenschaften und Methoden für den Zugriff auf Dokumente, die in einer XML-basierten Sprache abgefasst sind. Möglich ist eine solche Anwendung aber nur, weil alle diese Sprachen denselben, XML-bedingten syntaktischen Grundmustern folgen. Da die syntakti-**

schen Grundmuster von HTML denen von XHTML sehr ähnlich sind, ist das DOM in der Praxis auch auf HTML anwendbar. Doch der Wunsch ist verständlich, ein HTML zu haben, das sich exakt an die Syntax von XML hält.

- **Was für Script-Sprachen gilt, gilt ebenso auch für Stylesheet-Sprachen. Die selbst XML-basierte allgemeine Stylesheet-Sprache** XSL **(*Extensible Stylesheet Language = erweiterbare Formatsprache*) ist genau für diese Zwecke entworfen worden. Sie stellt eine einheitliche Grundlage für alle XML-basierten Sprachen zur Verfügung.**

XHTML ist also XML-gerechtes HTML. Der "Nachbau" von HTML 4.0 in XHTML 1.0 ist so weit gelungen, dass es in XHTML die gleichen Elemente, Attribute und Verschachtelungsregeln gibt wie in HTML. Systembedingt durch die Syntax von XML gibt es jedoch diverse Unterschiede im Detail, die Sie kennen müssen, wenn Sie Ihre Web-Seiten in XHTML statt in HTML schreiben wollen.

XHTML hat einen zwiespältigen Charakter, denn einerseits soll es alle Vorteile einer XML-Sprache bieten. Andererseits soll XHTML eine praktische anwendbare Sprache für Webseiten sein. XHTML wurde deshalb so angelegt, dass eine Kompatibilität mit Browsern, die lediglich auf HTML-Syntax ausgelegt sind oder nicht über die nötigen XML-Fähigkeiten verfügen, möglich ist. Tatsächlich verarbeiten die verbreiteten Browser XHTML-Dokumente und stellen sie genauso wie HTML-Dokumente am Bildschirm dar. Die Voraussetzung ist, dass Sie einige Unterschiede beim Schreiben des Codes beachten, die auf dieser Seite aufgelistet sind. Einzelne Probleme, besonders mit älteren Browsern, sind dennoch nicht auszuschließen.

Durch diese Kompatibilität zu HTML-Browsern ergeben sich zwei Arten der Verarbeitung von XHTML. Bei der Verarbeitung als HTML machen sich die Besonderheiten von XHTML beim Betrachten Ihrer Webseite nicht bemerkbar. Viele der Unterschiede zwischen HTML und XHTML kommen nämlich erst dann zum tragen, wenn das XHTML-Dokument nach den Regeln von XML verarbeitet wird. Ebenso entfalten sich die Vorteile von XHTML erst dann, wenn beim Erstellen, Ändern oder Lesen tatsächlich XML-Techniken genutzt werden. Neuere Browser, die XHTML korrekt verarbeiten und darstellen können, sollten ein XHTML-Dokument daher sinnvollerweise als XML verarbeiten. Der Abschnitt ↓ MIME-Typen erläutert, wie sich der Verarbeitungsmodus unter Berücksichtigung der Browserfähigkeiten bestimmen lässt.

Unterschied: MIME-Typen

Der MIME-Typ für gewöhnliche HTML-Dokumente lautet stets `text/html`. XHTML-Dokumente können diesen MIME-Typ ebenfalls haben, wenn sie die Empfehlungen des W3-Konsortiums zur Abwärtskompatibilität beachten. Gleichzeitig sind jedoch auch die allgemeinen MIME-Typen `text/xml` und `application/xml` erlaubt, die für alle von XML abgeleiteten Sprachen gedacht sind. Das W3-Konsortium hat zusätzlich den MIME-Typ `application/xhtml+xml` speziell für XHTML etabliert und empfiehlt, diesen neben `text/html` zu benutzen. Für das Erstellen von XHTML-Dokumenten hat dies direkt zunächst keine Bedeutung. Wohl aber für die Kommunikation zwischen Web-Browser und Web-Server. Wenn der Server dem Browser die angeforderte XHTML-Datei mit dem Typ `application/xhtml+xml` sendet, muss der Browser in der Lage sein, diesen MIME-Typ zu verarbeiten und Dateien dieses Typs am Bildschirm anzuzeigen. Anderenfalls kann es beispielsweise passieren, dass der Browser dem Anwender die Datei zum Download anbietet, statt sie ins Anzeigefenster zu laden. Das W3-Konsortium hat eine Übersicht über die Browserunterstützung der verschiedenen MIME-Typen zusammengestellt.

In der Praxis hat es sich durchgesetzt, XHTML-Dokumente mit dem MIME-Typ `text/html` auszuliefern. Nur in den Fällen, in denen der Browser dem Server bei der HTTP-Anfrage über die `Accept`-Angabe ausdrücklich mitteilt, dass er `application/xhtml+xml` verarbeiten kann, sollte die Datei mit diesem MIME-Typ gesendet werden. Auf der Seite Mit dem Apache XHTML-Seiten ausliefern werden Möglichkeiten vorgestellt, wie Sie XHTML-Dateien je nach Fähigkeiten des verwendeten Browsern entweder mit `text/html` oder `application/xhtml+xml` ausliefern können.

Der Unterschied der MIME-Typen ist kein rein theoretischer, denn der MIME-Typ bestimmt die Art der Verarbeitung des XHTML-Dokuments. Bei `text/html` nutzen die Browser den herkömmlichen HTML-Parser bei der Verarbeitung des Quellcodes. Die heute üblichen HTML-Parser der Browser lassen syntaktische Fehler im Dokument einfach durchgehen und versuchen das Dokument "irgendwie" anzuzeigen. Bei `application/xhtml+xml` behandelt ein Browser, der den MIME-Typ korrekt verarbeitet, die XHTML-Datei als echtes XML-Dokument und benutzt daher seinen XML-Parser. Der XML-Parser arbeitet im Gegensatz zum HTML-Parser nach festen, vorgegebenen Regeln. Er erwartet, dass das Dokument die strengen Syntaxregeln von XML einhält. Wenn er auf einen syntaktischen Fehler stößt, muss er die Verarbeitung abbrechen. Im Anzeigefenster des Browsers erscheint dann entweder nur eine Fehlermeldung anstelle des Dokuments, oder, wie bei vielen aktuellen Browsern, das Dokument nur bis zur Stelle des Fehlers. Wenn Sie also den MIME-Typ `application/xhtml+xml` verwenden, müssen sie peinlichst darauf achten, dass Ihre XHTML-Dateien wohlgeformt sind. Dies können Sie beispielsweise mit dem Validator des W3-Konsortiums prüfen.

Wohlgeformtheit eines XML-Dokuments

Der Begriff "Wohlgeformtheit", der Ihnen im Zusammenhang mit XML immer wieder begegnen wird, bedeutet, dass eine Datei die Regeln von XML korrekt einhält.

Beispiel einer wohlgeformten XML-Datei:

```
<?xml version="1.0"?>
        <Dialog>
          <Adam Emotion="heftig">ich liebe dich!</Adam>
          <Eva Emotion="heftig">ich dich auch!</Eva>
        </Dialog>
```

Erläuterung:
Es handelt sich aus folgenden Gründen um eine wohlgeformte XML-Datei:

- **am Beginn steht die XML-Deklaration, die den Bezug zu XML herstellt.**
- **Es gibt mindestens ein Datenelement (im Beispiel sind es drei: <Dialog>...</Dialog>, <Adam>...</Adam> und <Eva>...</Eva>)**
- **Es gibt bei den Datenelementen ein äußerstes Element (auch als Dokument-Element bezeichnet), das alle anderen Datenelemente enthält (im Beispiel: <Dialog>...</Dialog>).**

Was jedoch fehlt, ist der Bezug zu einer DTD, in der die verwendeten Elemente definiert sind. Deshalb kann man über das Beispiel sagen, es ist wohlgeformt, aber nicht gültig.

Gültigkeit eines XML-Dokuments / vollständiges XML-Dokument

Ein gültiges XML-Dokument enthält zu Beginn eine XML-Deklaration und anschließend eine Dokumenttyp-Deklaration, in der entweder eine externe DTD-Datei oder intern die nötigen DTD-Regeln angegeben werden. Man kann also sagen, ein gültiges XML-Dokument ist wohlgeformt und außerdem noch validierbar, also gegen die in einer DTD definierten Regeln überprüfbar.

Beispiel einer gültigen und vollständigen XML-Datei:

```
<?xml version="1.0"?>
        <!DOCTYPE quelle [
                <!ELEMENT quelle (adresse, beschreibung)>
                <!ELEMENT adresse (#PCDATA)>
                <!ELEMENT beschreibung (#PCDATA)>
        ]>
        <quelle>
                <adresse>http://www.willy-online.de/</adresse>
                 <beschreibung>alles zum wichtigsten Teil am Manne</beschreibung>
        </quelle>
```

Erläuterung:
Das Beispiel startet mit der üblichen XML-Deklaration. Danach folgt eine Dokumenttyp-Deklaration mit internen DTD-Regeln. Genausogut könnte an dieser Stelle auch eine Dokumenttyp-Deklaration stehen, die auf eine externe DTD verweist. Definiert wird im Beispiel ein Dokumenttyp namens quelle mit dem gleichnamigen Dokument-Element quelle und zwei davon abhängigen Elementen adresse und beschreibung. Zur genauen Syntax von DTD-Definitionen siehe Dokumenttyp-Definitionen (DTDs).
Im Anschluss an die Dokumenttyp-Deklaration folgen die Daten. Der **Name des Dokumenttyps**, im Beispiel quelle, muss gleich dem Namen des **Dokument-Elements** sein. Das Dokument-Element ist das äußerste Element in der Hierarchie, dem alle anderen Elemente untergeordnet sind. Im Beispiel ist der gesamte Datenbereich in <quelle>...</quelle> eingeschlossen.
Innerhalb des Dokument-Elements werden die Daten nach den Regeln notiert, die von der DTD vorgegeben werden. Im Beispiel wurden im DTD-Bereich die beiden Elemente adresse und beschreibung als innere Elemente des Dokument-Elements quelle definiert. Im Datenbereich spiegelt sich das wider durch die Notationen <adresse>...</adresse> und <beschreibung>...</beschreibung>. Der DTD-Bereich im Beispiel erlaubt nichts anderes als die einmalige Verwendung dieser beiden Elemente.
Das Beispiel erfüllt damit alle Kriterien einer gültigen und vollständigen XML-Datei.

Unterschied: Dateinamen

Neuere, XML-fähige Browser (Internet Explorer ab Version 5.x, Netscape ab Version 6.x) behandeln Dateien je nach Systemkonfiguration abhängig von der Dateinamenerweiterung unterschiedlich, wenn sie sonst keine Hinweise (MIME-Typ) darauf haben, wie die Datei zu behandeln ist. Wenn Sie in einer Datei zwar alle Regeln von XHTML einhalten, die Datei aber mit den typischen Endungen *.htm* oder *.html* abspeichern, benutzen diese Browser ihre HTML-Parser. Speichern Sie die Datei dagegen mit einer anderen Endung ab, z.B. *.xhtml*, dann benutzen die Browser unter Umständen ihre XML-Parser. Dies wirkt sich genauso aus wie bei den ↑ verschiedenen MIME-Typen.

Bei der Darstellung XML-geparster, fehlerfreier XHTML-Dokumente ohne CSS oder andere Stylesheets reagieren die Browser unterschiedlich. Netscape 6.x zeigt das Dokument wie ein HTML-Dokument an, also mit den voreingestellten Darstellungen etwa für Überschriften, Textabsätze, Listen, Tabellen usw. Der Internet Explorer stellt solche Dokumente dagegen als reine Element-Baumstruktur dar. Erst wenn Sie die verwendeten Elemente mit Stylesheets formatieren, zeigt der Internet Explorer das Dokument in nicht-schematischer Form an.

Solange Sie also XHTML-Standard-konforme Dateien mit den HTML-typischen Endungen *.htm* oder *.html* abspeichern, erreichen Sie eine "normale" Darstellung im Browser, verhindern aber auch, dass die Dateien im Browser auf XML-Basis verarbeitet werden. Speichern Sie die Dateien dagegen unter einer anderen, dem Browser unbekannten oder mit dem MIME-Typ `application/xml` verknüpften Dateiendung ab, werden die Dateien auf XML-Basis verarbeitet - in diesem Fall dürfen sie jedoch keine syntaktischen Fehler enthalten. Wenn im Anzeigefenster des Internet Explorers etwas anderes als ein Strukturbaum erscheinen soll, müssen Sie außerdem Stylesheets verwenden, um die Elemente zu formatieren.

Der Unterschied hinsichtlich der Dateinamen ist mit dem der MIME-Typen verknüpft. Ein Web-Server unterscheidet eine HTML-Datei für gewöhnlich über dessen Dateinamenerweiterung von einer XHTML-Datei. Dementsprechend wählt er einen geeigneten MIME-Typ, also etwa `text/html`, `application/xml` oder `application/xhtml+xml`. Beim verbreiteten ≡ Web-Server Apache können Sie unter der Voraussetzung, dass der Server eine aktuelle Konfiguration besitzt, mit der Dateinamenerweiterung `.html` für MIME-Typ `text/html` arbeiten. Für `application/xhtml+xml` können Sie die Erweiterung `.xhtml` verwenden. Bei veralteten Konfigurationen müssen Sie hingegen damit rechnen, dass der Web-Server die Erweiterung `.xhtml` nicht erkennt und die Datei stattdessen als Nur-Text sendet.

Unterschied: Die XML-Deklaration und die Zeichenkodierung

Das allererste, was am Anfang einer XHTML-Datei notiert werden sollte, ist eine so genannte XML-Deklaration. Damit geben Sie an, dass die folgende Datei XML-gerechte Daten enthält und welche Zeichenkodierung die Datei verwendet (siehe auch Computer und geschriebene Sprache). Eine solche Deklaration gibt es in HTML 4.0 nicht, sie ist also XHTML-spezifisch.

Beispiel für XHTML 1.0:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
```

Erläuterung:

Notieren Sie diese Angabe so wie in dem Beispiel gezeigt in der ersten Zeile einer XHTML-Datei. Derzeit ist beim Attribut `version` der Wert `1.0` sinnvoll. Zwar gibt es bereits Version 1.1 des XML-Standards, Sie sollten jedoch `1.0` angeben, sofern Sie nicht aus bestimmten Gründen XML 1.1 verwenden wollen.

Beim Attribut `encoding` können Sie die Zeichenkodierung angeben, nach der die die XHTML-Datei zu verarbeiten ist.
`encoding="ISO-8859-1"` steht für die Kodierung ISO-8859-1 (Latin-1) für westeuropäische Sprachen. Ein Zeichen belegt in der Datei immer ein Byte.
`encoding="UTF-8"` steht für eine internationale Kodierung auf Basis der ISO/IEC-10646-Norm (Unicode). Ein Zeichen belegt in der Datei eine variable Breite, z.B. ein Byte, aber auch zwei, drei oder vier Bytes.
`encoding="UTF-16"` steht ebenfalls für eine internationale Kodierung auf Basis der ISO/IEC-10646-Norm (Unicode). Ein Zeichen belegt in der Datei je nach Unicode-Position entweder zwei oder vier Bytes.

Beachten Sie:

Die Angabe einer XML-Deklaration ist nicht grundsätzlich notwendig, aber in Fällen äußerst ratsam. Denn bei der Verarbeitung als XML kommt der Angabe zur Zeichenkodierung eine besondere Wichtigkeit zu. Wenn Sie nämlich keine solche Angabe machen, nehmen XML-Parser per Voreinstellung die Unicode-Kodierung UTF-8 an (siehe dazu Standard-Zeichenkodierung und Unicode-Unterstützung). Wenn die Kodierung Ihres Dokuments von UTF-8 abweicht, sollten Sie unbedingt eine XML-Deklaration verwenden, um die verwendete Kodierung anzugeben. Ansonsten bricht ein XML-Parser die Verarbeitung mit einer Fehlermeldung ab. Es ist zwar möglich, die Kodierung der XHTML-Datei dem Web-Browser über die Kopfzeilen der HTTP-Antwort mitzuteilen. Dies erfordert allerdings eine spezielle Konfiguration des Web-Servers. Außerdem ist eine solche Angabe in der Regel nutzlos, wenn der Anwender das Dokument auf seiner Festplatte speichert und ansieht.

Wenn das Dokument hingegen als HTML verarbeitet wird, kann die XML-Deklaration die Dokumenttyp-Weiche des Internet Explorers negativ beeinflussen, so dass Sie sie gegebenenfalls weglassen sollten. In jedem Fall sollten Sie eine Kodierungsangabe in einem Meta-Element im Kopf des Dokuments einfügen, damit das Dokument von HTML-Parsern korrekt verarbeitet werden kann.

Unterschied: Dokumenttyp-Deklaration

XHTML 1.0 und HTML 4.0 haben unterschiedliche Dokumenttyp-Deklarationen. Die Dokumenttyp-Deklaration ("Doctype") nimmt Bezug auf die DTD (Dokumenttyp-Definition) und Sprachversion, die Sie in der Datei verwenden und an die Sie sich halten wollen. Ein strenger Parser kann die Anzeige der Datei z.B. im Browser verhindern, wenn die Datei syntaktische Fehler enthält. Maßgeblich dafür, was ein syntaktischer Fehler ist, ist die DTD, auf die Sie mit der Dokumenttyp-Deklaration Bezug nehmen.

Beispiel für HTML 4.01:

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

Entsprechendes Beispiel für XHTML 1.0:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
       "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Erläuterung:

Die Beispiele zeigen, wie Dokumenttypen für HTML 4.01 und XHTML 1.0 für die HTML-Variante Strict notiert werden. Auch in XHTML 1.0 gibt es die drei Varianten Strict, Transitional und Frameset, genau wie in HTML 4.0. Die beiden anderen Varianten für XHTML lauten:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
       "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
       "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
```

Beachten Sie:

So wie oben notiert, kann ein strenger XML-Parser eine bestehende Internet-Verbindung erfordern und sich die DTD von der angegebenen Web-Adresse laden, um das Dokument gegen die darin formulierten Regeln zu prüfen. Sie können die DTDs in Ihr eigenes Web-Projekt integrieren. Angenommen, Sie haben eine Startdatei namens *index.htm* und die *xhtml1-strict.dtd* in einem eigenen Unterverzeichnis namens *html-dtd* abgelegt, dann können Sie in der *index.htm* notieren:

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

Sie können die erforderlichen DTDs für XHTML downloaden. Die Zip-Datei (31 KByte) enthält die DTDs für alle drei Varianten Strict, Transitional und Frameset sowie drei zusätzliche Dateien mit Definitionen für benannte Zeichen.

Unterschied: HTML-Wurzelelement mit Namensraumangabe

Das einleitende `<html>`-Tag hat in HTML meistens keine Attribute. In XHTML müssen Sie jedoch den Namensraum für XHTML explizit angeben.

Beispiel für HTML 4.01:

```
<html>
<!-- Inhalt der Datei -->
</html>
```

Entsprechendes Beispiel für XHTML 1.0:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<!-- Inhalt der Datei -->
</html>
```

Erläuterung:

Notieren Sie in XHTML im einleitenden `<html>`-Tag immer das Attribut `xmlns`, das einen XML-Namensraum bezeichnet, und weisen Sie ihm den Wert `http://www.w3.org/1999/xhtml` zu. Dies ist ein spezieller URI, der nicht dazu gedacht ist, eine aufrufbare Web-Adresse anzugeben, sondern lediglich das Schema der Web-Adressierung nutzt, um einen weltweit eindeutigen Namen für den Namensraum zu vergeben. Die Web-Adresse existiert zwar, weil das W3-Konsortium so freundlich war, damit die Häufung unnötiger Aufruffehler auf ihrem Server zu vermeiden - doch die Adresse bedeutet nichts anderes als ein beliebiger eindeutiger Name.

Unterschied: Strengeres Einhalten des HTML-Grundgerüsts

Wenn Sie in HTML die Elemente `html`, `head` und `body` weglassen, jedoch das Element `title` und ansonsten gültiges HTML notieren, dann ist es aus Sicht von HTML ein fehlerfreies und vollständiges Dokument. In XHTML besteht diese Freiheit nicht. Hier muss eine HTML-Datei zwingend das übliche Grundgerüst einhalten und ein `head`- und ein `body`-Element enthalten.

Beispiel für HTML 4.01:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<title>Titel</title>
<h1>Text</h1>
```

Entsprechendes Beispiel für XHTML 1.0:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
       "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Text</title>
<!-- gegebenenfalls andere Elemente im Kopfbereich -->
</head>
<body>
<h1>Text</h1>
</body>
</html>
```

Erläuterung:

Zwar ist die gezeigte Form, ein HTML-4.0-Dokument zu notieren, für die Praxis alles andere als empfehlenswert, aber zumindest ist sie nicht verkehrt. Bei einem XHTML-Dokument muss dagegen immer das vollständige Grundgerüst notiert werden.

Unterschied: Kleinschreibung

In HTML 4.0 ist es egal, ob Sie `<TABLE BORDER="0">`, `<TABLE border="0">` oder `<Table Border="0">` notieren. HTML unterscheidet bei Namen von HTML-Elementen und Attributnamen nicht zwischen Groß- und Kleinschreibung. Nur bei der Wertzuweisung an manche Attribute wird Groß-/Kleinschreibung unterschieden - aber auch nur im Hinblick etwa auf Script-Sprachen. Bei Attributen mit festen erlaubten Werten, wie etwa `align`, ist es egal, ob Sie `CENTER` oder `center` notieren. Nicht so bei XHTML. XML unterscheidet nämlich strikt zwischen Groß- und Kleinschreibung. Das bedeutet, `<TABLE>` ist etwas anderes als `<table>`. Für XHTML wurde festgelegt, dass **alle Elementnamen und Attributnamen klein geschrieben** werden. Das Gleiche gilt auch für die festen Wertzuweisungen wie `center`.

Beispiel für HTML 4.01:

```
<OBJECT DATA="Video.mpg" TYPE="video/mpeg" ALIGN="LEFT"></OBJECT>
```

Entsprechendes Beispiel für XHTML 1.0:

```
<object data="Video.mpg" type="video/mpeg" align="left"></object>
```

Erläuterung:

Schreiben Sie in XHTML immer alle Elementnamen wie `object` oder Attributnamen wie `data` oder `type` klein. Schreiben Sie Wertzuweisungen im Zweifelsfall ebenfalls immer klein, also beispielsweise bei `align="left"`. Nur dort, wo in einer Wertzuweisung die Großschreibung von bestimmten Buchstaben erforderlich ist, wie im Beispiel beim Dateinamen `Video.mpg`, können Sie den Wert so schreiben wie es erforderlich ist.

Unterschied: Leere Elemente

In HTML 4.0 gibt es diverse leere Elemente. Das sind Elemente ohne Inhalt. Das Abschluss-Tag ist deshalb verboten, weil die Elemente als inhaltsleer definiert sind. Beispiele:
`<img>`, `<br>`, `<input>`, `<hr>`.
In XML-basierten Sprachen, also auch in XHTML, müssen solche leeren Elemente gesondert gekennzeichnet werden.

Beispiel für HTML 4.01:

```
<p>Text mit<br>Zeilenumbruch</p>
<p><img src="bild.png" alt="ein Bild"></p>
```

Entsprechendes Beispiel für XHTML 1.0:

```
<p>Text mit<br />Zeilenumbruch</p>
<p><img src="bild.png" alt="ein Bild" /></p>
```

Erläuterung:

Notieren Sie unmittelbar vor der schließenden spitzen Klammer des alleinstehenden Tags einen Schrägstrich, so dass am Ende die Zeichenfolge `/>` steht. Vor dem Schrägstrich sollte ein Leerzeichen stehen, um die Funktionalität in alten Browsern zu gewährleisten. Alternativ dazu können Sie auch ein Element mit Anfangs- und End-Tag notieren, z.B. `<br></br>`. Dabei darf jedoch nichts, auch kein Leerzeichen und kein Zeilenumbruch, zwischen dem Anfangs- und dem End-Tag stehen, ansonsten ist es ungültiges XHTML.

Wenn Sie Elemente, die Inhalt haben können, ohne Inhalt verwenden, z.B. `<p></p>`, dann notieren Sie sie in XHTML jedoch besser in dieser Form und nicht in der Form `<p />`. Denn die verbreiteten HTML-Parser behandeln `<p />` wie `<p>`.

Unterschied: Elemente mit optionalem Abschluss-Tag

In HTML 4.0 gibt es diverse Elemente mit optionalem Abschluss-Tag. Beispiele:
`<body>`, `<td>`, `<dd>`, `<dt>`, `<option>`.
Notieren Sie solche Elemente in XHTML immer mit Anfangs- und Abschluss-Tag.

Beispiel für HTML 4.01:

```
<select name="Auswahl" size="1">
  <option>1. Eintrag
  <option>2. Eintrag
  <option>3. Eintrag
</select>
```

Entsprechendes Beispiel für XHTML 1.0:

```
<select name="Auswahl" size="1">
  <option>1. Eintrag</option>
  <option>2. Eintrag</option>
  <option>3. Eintrag</option>
</select>
```

Erläuterung:

Alle Elemente, die Inhalt haben können, müssen in XHTML mit Anfangs- und Abschluss-Tag gekennzeichnet werden.

Unterschied: Alleinstehende (leere) Attribute

In HTML 4.0 gibt es diverse Attribute, die keine Wertzuweisung erhalten. Beispiele:
`<input checked>`, `<textarea readonly>`, `<input disabled>`, `<select multiple>`, `<hr noshade>`, `<td nowrap>`, `<script defer>`.
In XML-basierten Sprachen dagegen muss allen Attributen ausdrücklich ein Wert zugewiesen werden. Da diese Attribute nur einen möglichen Wert annehmen können, hat man sich darauf verständigt, als Wert einfach den Attributnamen zu nehmen.

Beispiel für HTML 4.01:

```
<td nowrap>Inhalt</td>
<p>Text mit<hr noshade>Trennlinie</p>
```

Entsprechendes Beispiel für XHTML 1.0:

```
<td nowrap="nowrap">Inhalt</td>
<p>Text mit<hr noshade="noshade" />Trennlinie</p>
```

Erläuterung:

Weisen Sie leeren Attributen in XHTML einfach ihren eigenen Namen als Wert zu. Bei den meisten bekannten Browsern gibt es damit keine Probleme.

Unterschied: Leerraum in Wertzuweisungen

Bei Wertzuweisungen an Attribute müssen Sie in XHTML besser aufpassen als in HTML. Leerzeichen sind erlaubt wo erforderlich, doch Zeilenumbrüche sollten Sie vermeiden.

Beispiel für empfohlenes XHTML 1.0:

```
<p title="Anfang der Geschichte">Text Text Text</p>
```

Beispiel für problematisches XHTML 1.0:

```
<p title="Anfang
der
Geschichte">Text Text Text</p>
```

Erläuterung:

Sie sollten sich nicht darauf verlassen, dass die Browser mit Zeilenumbrüchen in Attributwerten so umgehen, wie Sie es erwarten. Verwenden Sie möglichst keine Zeilenumbrüche.

Unterschied: Verweise zu Ankern

In HTML können Sie Verweise zu Ankern notieren, z.B. `<a href="#anker">Verweis</a>`. Der entsprechende Anker ist dann notiert mit `<a name="anker">irgendwas</a>`. Auch andere Attribute verwenden diese Technik, wie etwa das `usemap`-Attribut beim `img`-Element zum Verweis auf ein `map`-Element, bei dem ein entsprechendes `name`-Attribut notiert ist.

In XHTML funktionieren solche Verweise nicht. XML benötigt Namen vom Typ "ID", also dokumentweit eindeutige Bezeichner, um solche Beziehungen auflösen zu können. In XHTML können Sie das Universalattribut `id` verwenden, um solche Verweise zu realisieren. Es ist zwar bereits in HTML 4 möglich, einen Anker alleine mit dem `id`-Attribut zu realisieren, z.B. `<h2 id="anker">&Uuml;berschrift</h2>`, aber dennoch interpretieren manche alte Browser wie Netscape 4 nur `<a name="anker">irgendwas</a>` als Anker. Sie sollten daher beide Attribute notieren, damit sowohl ältere als auch neueren Browser den Anker verstehen und der Anker auch dannn funktioniert, wenn der Browser den XHTML-Verarbeitungsmodus einsetzt.

Beispiel für HTML 4.01:

```
<a href="aAnker">Verweis</a>
<p>viel Inhalt</p>
<a name="anker">irgendwas</a>
```

Entsprechendes Beispiel für XHTML 1.0:

```
<a href="#Anker">Verweis</a>
<p>viel Inhalt</p>
<a id="anker" name="anker">irgendwas</a>
```

Entsprechendes Beispiel für XHTML 1.1:

```
<a href="#Anker">Verweis</a>
<p>viel Inhalt</p>
<a id="anker">irgendwas</a>
```

Erläuterung:

Notieren Sie in dem `a`-Element für den Anker beide Attribute `id` und `name`. Weisen Sie beiden Attributen exakt den gleichen Namen zu. Ein Browser, der den Verweis nach XHTML-Logik ausführt, benötigt das `id`-Attribut, während ein Browser, der den Verweis nach HTML-Logik ausführt, das `name`-Attribut zur Ermittlung des Ankers verwendet. In XHTML 1.1 entfällt das `name`-Attribut.

Unterschied: Das Universalattribut lang

XML-basierte Sprachen benutzen normalerweise anstelle von `lang` das XML-Universalattribut `xml:lang`. Da das `lang`-Attribut in XHTML jedoch ebenfalls zur Verfügung steht, existieren also zwei Attribute für die gleiche Sache. XHTML-Parser bevorzugen im Zweifelsfall das Attribut `xml:lang`, das aber von reinen HTML-Parsern nicht erkannt wird. Notieren Sie deshalb in der Praxis stets beide Varianten, falls Sie das Attribut verwenden.

Beispiel für HTML 4.01:

```
<html lang="de">
<!-- Inhalt der Datei -->
</html>
```

Entsprechendes Beispiel für XHTML 1.0:

```
<html xmlns="http://www.w3.org/1999/xhtml" lang="de" xml:lang="de">
<!-- Inhalt der Datei -->
</html>
```

Entsprechendes Beispiel für XHTML 1.1:

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="de">
<!-- Inhalt der Datei -->
</html>
```

Erläuterung:

Die Sprache des Dokuments wird durch die Angabe `lang="de"` im `html`-Element dokumentweit festgelegt. Im XHTML 1.0-Beispiel kommt, neben der ↑ Namensraumangabe, zusätzlich das Attribut `xml:lang="de"` hinzu. Die Reihenfolge der Attribute spielt keine Rolle. In XHTML 1.1 entfällt das `lang`-Attribut zugunsten von `xml:lang`.

Unterschied: Inhalt von Script- und Style-Bereichen

In HTML 4.0 ist der Inhalt der Elemente script und style als "CDATA" festgelegt, während er in XHTML als "PCDATA" behandelt wird. Im Klartext bedeutet das, dass bei HTML 4.0 innerhalb eines Script-Bereichs oder Style-Bereichs HTML-eigene Zeichen wie <, >, & und " nicht maskiert werden müssen. Nicht so in XHTML. Wenn Sie beispielsweise in einem JavaScript innerhalb einer Bedingung vergleichen, ob ein Wert kleiner oder größer als ein anderer Wert ist, müssen Sie die spitzen Klammern außerhalb einer Zeichenkette notieren und können sie daher an der Stelle auch nicht maskieren. Um das Problem zu lösen, können Sie das Script in einen CDATA-Abschnitt anschließen. Dies ist eine XML-typische Notation.

Da XML-Parser HTML-Kommentare (und damit auch den enthaltenen JavaScript-Code) entfernen dürfen, sollte auf die früher übliche Auskommentierung des Scripts verzichtet werden, die für die heute verwendeten Browser auch nicht mehr erforderlich ist.

Beispiel für HTML 4.01:

```
<script type="text/javascript">
<!--
  /* JavaScript-Kommentar: jetzt folgt ein kleines Script */
  if (parseInt(navigator.appVersion) < 5)
    alert("Oh, ein sehr alter Browser");
  /* und damit ist das Script auch schon zu ende */
//-->
</script>
```

Entsprechendes Beispiel für XHTML 1.0:

```
<script type="text/javascript">
/* <![CDATA[ */
  /* JavaScript-Kommentar: jetzt folgt ein kleines Script */
  if (parseInt(navigator.appVersion) < 5)
    alert("Oh, ein sehr alter Browser");
  /* und damit ist das Script auch schon zu ende */
/* ]]> */
</script>
```

Erläuterung:

Der "CDATA"-Abschnitt beginnt mit `<![CDATA[` und endet mit `]]>`. Ohne diese Umklammerung würde ein XML-Parser das <-Zeichen innerhalb des Scripts als Fehler ankreiden. Damit der Browser die Anfang- und Endmarkierung des CDATA-Abschnitts nicht als JavaScript-Code zu interpretieren versucht, werden sie durch JavaScript-Kommentare vor ihm versteckt. Den Inhalt von `style`-Elementen können Sie ebenso umschließen und analog die Markierungen mit CSS-Kommentaren vor dem CSS-Parser verbergen.

Beachten Sie:

Um Fehler bei JavaScript-Interpretern zu vermeiden, bleibt die Möglichkeit, JavaScript in separaten Dateien zu notieren. Auch bei Stylesheets besteht die Möglichkeit, Formate zentral in einer separaten CSS-Datei zu definieren.

Unterschied: Ausnahmen von Verschachtelungsregeln

In der Dokumenttyp-Definition von HTML 4.0 sind einige Ausnahmen bei Verschachtelungsregeln für Elemente definiert. Dies ist beispielsweise bei dem a-Element nötig, das kein anderes a-Element enthalten darf. Die DTD kann zwar festgelegen, dass ein a-Element nicht direkt ein a-Element enthalten darf, so dass `<a><a>...</a></a>` nicht erlaubt ist. Ein a-Element darf jedoch andere Elemente enhalten, die ihrerseits ein a-Element enthalten dürfen, z.B. `span`. In diesem Fall greifen die normalen Regeln nicht und es ist eine Ausnahmeregel nötig, um z.B. `<a><span><a>...</a></span></a>` zu verbieten.

Mit Hilfe von SGML ist die Formulierung solcher Ausnahmen möglich. So ist beispielsweise festgelegt, dass ein a-Element auch als indirektes Kindelement kein anderes a-Element enthalten darf. XML dagegen bietet keine Möglichkeit, solche Ausnahmen zu formulieren. In den XML-basierten Dokumenttyp-Definitionen von XHTML fehlen solche Verschachtelungsverbote daher. Somit ist z.B. `<a><a>...</a></a>` gemäß den Regeln der DTDs von XHTML zunächst erlaubt. Um eine möglichst hunderprozentige Kompatibilität zwischen HTML und XHTML zu erreichen, blieb dem W3-Konsortium nur der Ausweg, diese Verschachtelungsverbote verbal als "normativen Anhang" zu den DTDs zu erklären.

Folgende Verschachtelungsregeln sind betroffen:

- **`a`-Elemente dürfen keine weiteren `a`-Elemente enthalten.**
- **`pre`-Elemente dürfen keine Elemente `img`, `object`, `big`, `small`, `sub` oder `sup` enthalten.**
- **`button`-Elemente dürfen keine Elemente `input`, `select`, `textarea`, `label`, `button`, `form`, `fieldset`, `iframe` oder `isindex` enthalten.**
- **`label`-Elemente dürfen keine weiteren `label`-Elemente enthalten.**
- **`form`-Elemente dürfen keine weiteren `form`-Elemente enthalten.**

Beachten Sie:

Wenn Sie Ihr XHTML-Dokument z.B. mit dem W3C-Validator auf syntaktische Korrektheit prüfen, werden solche Verschachtelungsfehler nicht gefunden, denn dieser Validator prüft das Dokument unter Berücksichtigung der Dokumenttyp-Definition. Alternativ können Sie den XHTML 1.0 Schema-Validator verwenden, der neben anderen strengeren Tests die Einhaltung der genannten Verschachtelungsregeln überprüft.

Beispiel eines XHTML-Dokuments

Das folgende Beispiel zeigt ein vollständiges XHTML-Dokument, in dem Sie die Unterschiede zu HTML noch einmal im Zusammenhang sehen können. Sie können das zugehörige Anzeigebeispiel einmal als Datei mit der Endung *.htm*, einmal als Datei mit der Endung *.xhtml* und einmal als Datei mit der Endung *.xml* aufrufen, um selber zu testen, was in Ihrem Browser dann passiert. Es handelt sich um exakt den gleichen Inhalt, nur die Dateinamen sind unterschiedlich.

Beispiel:

Anzeigebeispiel: So sieht's aus (*beispiel**.htm***)
Anzeigebeispiel: So sieht's aus (*beispiel**.xhtml***)
Anzeigebeispiel: So sieht's aus (*beispiel**.xml***)

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />
<title>Wenn HTML zu XHTML wird</title>
</head>
<body>

<h1><a name="start" id="start">Wenn HTML zu XHTML wird</a></h1>

<p>Dann m&uuml;ssen alle Elemente mit Inhalt ein End-Tag haben.</p>

<p>Leere Elemente<br />m&uuml;ssen einen Schr&auml;gstrich am Ende haben.</p>

<hr noshade="noshade" />
<p>Leere Attribute erhalten ihren eigenen Namen als Wert zugewiesen.</p>

<p><a href="#start">Verweise zu Ankern</a> springen zum Zielelement
aufgrund des id-Attributs, nicht das name-Attributs.</p>

<script type="text/javascript" src="zeitstempel.js"></script>
<!-- So werden Scripts am sichersten eingebunden. -->

</body>
</html>
```

Erläuterung:

Bei der Dokumenttyp-Deklaration ist nur `xhtml1-transitional.dtd` als URI notiert, ohne weitere Web-Adresse oder Pfadangabe. Das liegt daran, dass die DTD mit dem Namen *xhtml1-transitional.dtd* im gleichen Verzeichnis wie das Dokument selber liegt.