

XML-Syntax (SYN)

Lernziele

- Sie verstehen den Jargon der XML-Spezifikation.
- Sie kennen die typischen Bestandteile eines XML-Dokuments.
- Sie kennen die logische und physikalische Struktur eines XML-Dokuments.
- Sie sind sicher im Umgang mit den Regeln zur Erstellung von XML-Dokumenten.
- Sie wissen, welche Zeichen im Markup, XML-Namen und Zeichendaten zulässig sind.
- Sie setzen das erworbene Wissen praktisch in den Übungen ein!

XML-Terminologie

XML-Dokumente sind eine Klasse von Datenobjekten, um die sich bei XML alles dreht. **XML-Prozessoren** lesen den Inhalt von XML-Dokumenten und stellen ihn weiterverarbeitenden **Anwendungen** zur Verfügung. XML-Prozessoren werden auch **XML-Parser** genannt.

Ein XML-Dokument ist **wohlgeformt (*well-formed*)**, wenn es den Regeln der W3C-Empfehlung entspricht. Erfüllt es darüber hinaus noch die Regeln einer im zugeordneten DTD, dann ist es **gültig (*valid*)**.

Man unterscheidet zwei Arten von XML-Prozessoren:

- validierende XML-Prozessoren
- nicht-validierende XML-Prozessoren

Nicht validierende XML-Prozessoren prüfen XML-Dokumente lediglich auf Wohlgeformtheit, validierende **XML-Prozessoren** darüber hinaus auf Gültigkeit.

Ein XML-Dokument speichert Daten aus reinem **Text**. Text ist entweder **Markup** oder **Zeichendaten (*character data*)**.

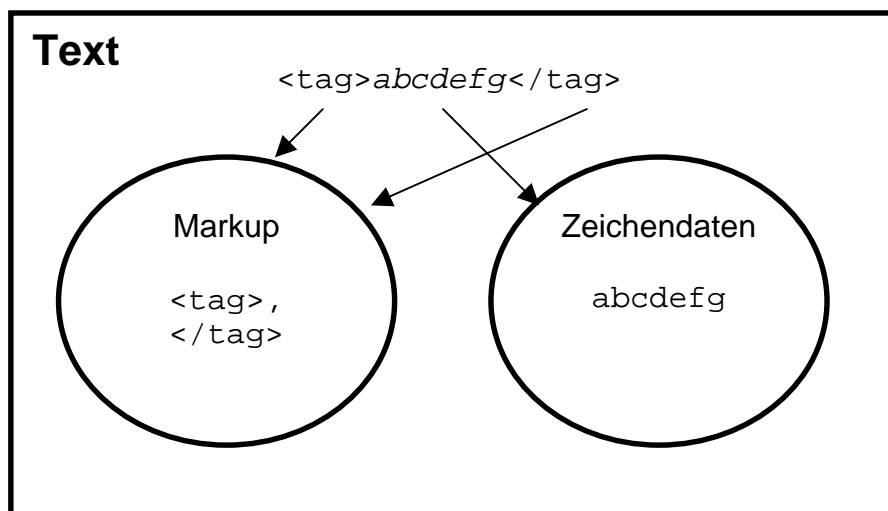


Abbildung SYN-1: Text, Markup und Zeichendaten

URIs, URLs und URNs

URI = Uniform Resource Identifier
URL = Uniform Resource Locator
URN = Uniform Resource Name

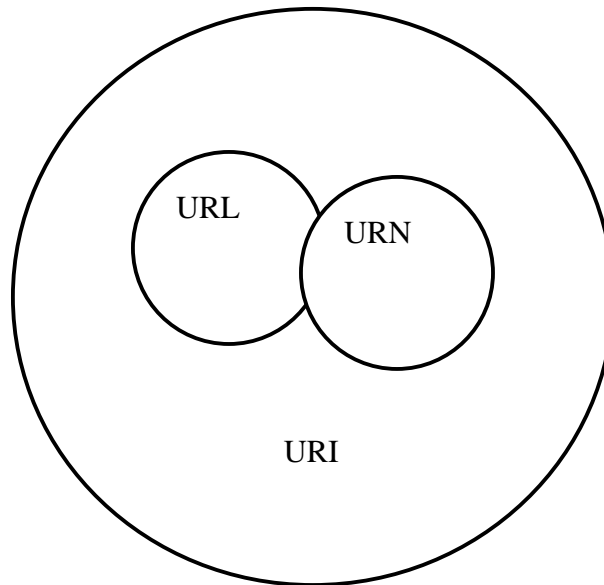


Abbildung SYN-2: Verhältnis URI, URN und URL

URIs sind Zeichenketten, die eine abstrakte oder physikalische Ressource identifizieren. **URLs** und **URNs** sind spezifische Ausprägungen von URIs. URLs sind als Adressierungsschemata im WWW bekannt. **URLs** identifizieren Ressourcen über die Zugriffsmethode und den Pfad dorthin (z.B.: *http://www.meinServer.de/index.html*), d.h. sie *lokalisieren* die Ressource. **URNs** identifizieren Ressourcen über einen dauerhaften (persistenten) und weltweit eindeutigen *Namen*. Der Name bleibt dabei unabhängig von der Existenz der Ressource eindeutig und dauerhaft (z.B.: *urn:nbn:de:gbr:089-3304534323*).

Elementares Markup

Elemente

Elemente sind die Hauptbausteine von XML-Dokumenten. Sie sind eine Art Container für den eigentlichen Inhalt des Dokuments.

Man unterscheidet:

- Elemente mit Inhalt
- Elemente ohne Inhalt (**leere Elemente**)

Element mit Inhalt

Element = Start-Tag + Inhalt + End-Tag

Beispiel SYN-1: Element

```
<elementtypName>Inhalt</elementtypName>
```

Jedes Element gehört zu einem **Elementtyp**, der durch einen **Namen** identifiziert wird. Der Name wird auch **Generic Identifier** genannt.

Ein Element mit Inhalt besteht also aus zwei Tags, dem öffnenden **Start-Tag**, das den Namen des Element(-typs) zwischen einem Kleiner-Als-Zeichen(<) und einem Größer-Als-Zeichen(>) einschließt, und einem schließenden **End-Tag**, das bis auf einen Querstrich (/), der vor dem Elementnamen erscheint, mit dem Start-Tag identisch ist.

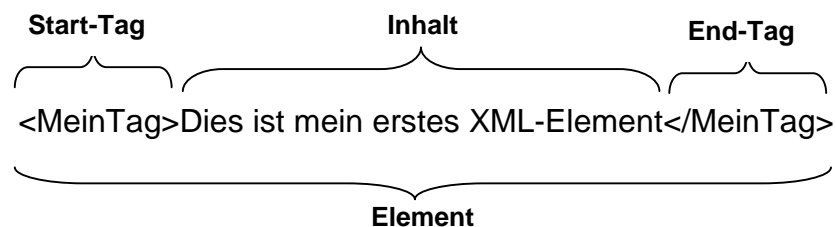


Abbildung SYN-3: Tags und Elemente

Leeres Element 1

Element (leer) = Start-Tag + End-Tag

Beispiel SYN-2: Leeres Element 1

```
<leeresElement></leeresElement>
```

oder

Leeres Element 2

Element (leer) = Leeres-Element-Tag

Beispiel SYN-3: Leeres Element 2

```
<leeresElement/>
```

Bei einem leeren Element steht kein Inhalt (Text oder andere Elemente) zwischen dem Start-Tag und dem End-Tag (auch kein Leerzeichen!). Beide können daher zu einem Tag zusammengezogen werden. Dabei setzt man einfach einen Schrägstrich (*Slash*) vor das schließende Größer-Als-Zeichen.

Leere Elemente werden oft benutzt, um einem Dokument nicht-textuellen Inhalt wie z.B. Bilder hinzuzufügen.

Beispiel SYN-4: Anwendungsfall leere Elemente

```
<bild quelle="logo.gif"></bild>
```

```
<bild quelle="logo.gif"/>
```

Wie obige Beispiele zeigen, können leere Elemente (natürlich auch Elemente mit Inhalt) Attribute besitzen.

Attribute

Elemente können ein oder mehrere **Attribute** besitzen. Attribute enthalten Informationen über Elemente oder Elementinhalte. Sie verhalten sich zu Elementen ähnlich wie Adjektive zu Substantiven. Attribute stehen immer im Start-Tag des jeweiligen Elements.

Attribut

Attribut = Attribut-Name + Attribut-Wert

Beispiel SYN-5: Attribut

```
<elementname Attribut-Name="Attribut-Wert">  
  Inhalt  
</elementname>
```

Der Attribut-Name ist durch ein Leerzeichen (*White Space*) vom vorhergehenden Text zu trennen. Der Attributwert steht nach einem Gleichheitszeichen (=) in doppelten oder einfachen Anführungszeichen.

Beispiel SYN-6: Verschiedene Elemente mit Attributen

```
<bild quelle="logo.gif">
```

```
<preis währung='DM'>24,50</preis>
```

```
<artikel zeitschrift="Spiegel" seitenzahl="3">  
  <autor>Rudolf Augstein</autor>  
  <titel>Bismarck und seine Gefährten</titel>  
</artikel>
```

Unterschiede zu HTML-Regeln

XML-Prozessoren sind streng, was die Auslegung von Regeln angeht. Anders als die meisten HTML-Browser quittieren sie fehlerhafte Syntax gnadenlos mit Fehlermeldungen. Wer von HTML kommt, sollte insbesondere folgende Dinge beachten:

- Attribut-Werte *müssen* in Anführungszeichen stehen.
- Jedes nicht leere Element muss durch ein End-Tag geschlossen werden.
- Tags müssen korrekt geschachtelt sein.
- Groß- und Kleinschreibung ist relevant (*Case-Sensitivität*).

Entity-Referenzen

Aus physikalischer Sicht bestehen XML-Dokumente aus **Entities**. Ein Entity ist laut XML-Spezifikation eine Speichereinheit. Im einfachsten Fall also eine Datei oder auch ein Datenobjekt. Entities sind somit die Informationseinheiten, aus denen sich ein XML-Dokument zusammensetzt. Das XML-Dokument selbst wird physikalisch als **Dokument-Entity (*document entity*)** bezeichnet.

Entities werden durch ihren **Namen** referenziert. Die eigentlichen Daten werden - in Analogie zu den Elementen - als **Inhalt (*content*)** bezeichnet.

Vereinfacht gesagt sind **Entity-Referenzen (*entity reference*)** Abkürzungen für andere Texte oder Datenobjekte. Dabei kann es sich um kurze Zeichenfolgen, aber auch ganze Dateien handeln.

Entity-Referenz

Entity-Referenz = & + Name + ;

Beispiel SYN-7: Entity-Referenz

```
<!-- Entity-Deklaration in der DTD -->

<!ENTITY rechte "Alle Rechte vorbehalten">

<!-- Entity-Referenz im XML-Dokument -->

<?xml version="1.0"?>
<impressum>
```

<copyright>(c) 2000, &rechte;</copyright>
</impressum>

Zeichendaten

Zeichendaten bestehen aus einzelnen **Zeichen (characters)**.

Erlaubte Zeichen (Zeichensatz)

XML-Dokumente verwenden als Zeichensatz **Unicode** (ISO/IEC 10646). Es handelt sich um einen sogenannten *16-Bit-Zeichensatz* (ASCII = 8 Bit). Damit lassen sich 65.536 ($=2^{16}$) Zeichen darstellen, was im Gegensatz zu ASCII für die Zeichen (fast) aller modernen Sprachen ausreicht.

TOOL-TIPP:

*Windows NT und 2000 liefern Ihnen mit dem **Notepad** einen Unicode-fähigen Texteditor mit. Geben Sie einfach im *Save-As-Dialog* bei *Encoding Unicode* an. Sie können jedoch auch jeden beliebigen Texteditor zur Erstellung von XML-Dateien benutzen.*

Zeichendaten können fast alle Zeichen des Unicode-Zeichensatzes (Version 3) beinhalten, außer den sogenannten Surrogat-Blöcken.

Gültige XML-Zeichen:

Code (hex)	Code (dez)	Beschreibung
09	09	ASCII-Steuerzeichen - horizontaler Tabulator (<i>Horizontal tab</i>) (HT)
0A	10	ASCII-Steuerzeichen - Zeilenvorschub (<i>Line-feed</i>) (LF)
0D	13	ASCII-Steuerzeichen - Wagenrücklauf (<i>Carriage return</i>) (CR)
20 - 7F	20 - 127	ASCII-Zeichen (lateinische Basiszeichen)
80 - D7FF	128 - 55295	Andere lateinische und nicht-lateinische Zeichen, Symbole usw.
E000 - FFFD	57344 - 65533	Private Use Area, zusätzliche CJKs (Chinese-Japanese-Korean), Ligaturen, andere Präsentationsformen und Unicode Specials
10000 - 10FFFF	65536 - 1114111	Unicode-Surrogat-Äquivalente und High Private Use Area

Der Unicode-Standard stellt mit den sogenannten **Private Use Areas** Blöcke für anwendungsspezifische Zeichen zur Verfügung. Die Nutzung dieser Blöcke sollten Sie vermeiden, wenn Ihre XML-Dokumente zum Datenaustausch gedacht sind.

Ungültige XML-Zeichen:

Code (hex)	Code (dez)	Beschreibung
00 - 08	00 - 08	ASCII C0 Steuerzeichen
0B - 0C	11 - 12	ASCII C0 Steuerzeichen
0E - 0F	14 - 19	ASCII C0 Steuerzeichen
D800 - DB7F	55296 - 56191	Unicode High Surrogates
DB80 - DBFF	56192 - 56319	Unicode High Private Use Surrogates
DC00 - DFFF	56320 - 57343	Unicode Low Surrogates
FFFE	65534	Byte-order Mark (BOM)
FFFF	65535	kein Zeichen
110000 - FFFFFFFF	1114112 - 4294967295	Werte sind reserviert für Versionen nach Unicode 3.0

Ausführliche Informationen zum Unicode-Standard finden Sie auf den Internet-Seiten des **Unicode Consortium**: <http://www.unicode.org>.

Zeichenkodierung

- Alle XML-Prozessoren müssen die Kodierungen **UTF-8** und **UTF-16** unterstützen.
- Ein XML-Prozessor unterscheidet zwischen UTF-8 und UTF-16 aufgrund der sogenannten **Byte-Order-Markierung (BOM)**, mit der UTF-16-Datenströme beginnen. Die BOM hat die hexadezimale Kodierung `FFFE`.

Hinweis:

ASCII ist eine Teilmenge von UTF-8

Markup-Zeichen

Folgende Zeichen sind Markup und müssen/sollten maskiert werden, wenn Sie in Zeichendaten auftauchen sollen:

Markup-Zeichen

Zeichen	Code (hex)	Code (dez)	Beschreibung
&	26	38	kaufmännisches Und
<	3C	60	Kleiner-Als-Zeichen
>	3E	62	Größer-Als-Zeichen
'	27	39	Apostroph
"	22	34	Anführungszeichen

Beachten Sie:

Strenggenommen müssen Sie bei Zeichendaten, die als Elementinhalt auftauchen, nur das kaufmännische Und (&) und das Kleiner-Als-Zeichen (<) maskieren und bei der Zeichenfolge]]> das Größer-Als-Zeichen markieren. Wollen Sie allerdings kompatibel zu SGML bleiben müssen Sie das Größer-Als-Zeichen immer maskieren. Apostroph- und Anführungszeichen-Maskierung wird nur in Attributwerten benötigt (s. auch Abschnitt *Entity-Referenzen*).

Zeichenreferenzen

Zeichenreferenzen (***character references***) dienen dazu, Zeichen einzugeben,

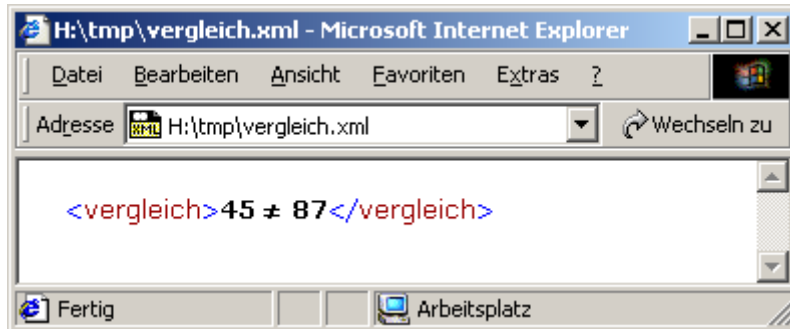
- die nicht auf der Tastatur verfügbar sind
- die von der Codierung des XML-Dokuments unabhängig sein sollen.

Sie enthalten die Nummer (in dezimaler Notation) des Zeichens im Unicode-Zeichensatz. Diese Nummer wird zwischen &# und ; eingeschlossen. Falls Sie hexadezimale Notation vorziehen, schließen Sie die Zahl zwischen &#x und ; ein.

Beispiel SYN-8: Zeichenreferenz für ≠

```
<vergleich>45 &#x2260; 87</vergleich>
```

Ausgabe:



Literale

Ein **Literal** ist eine beliebige, in Anführungszeichen eingeschlossene Zeichenkette, die nicht das Anführungszeichen enthält, das zur Begrenzung der Zeichenkette verwendet wird.

Literale werden verwendet zur Angabe von

- Attributwerten
- Inhalten von (internen) Entities
- System-Identifiern und Public-Identifiern von externen Entities und Dokumenttyp-Deklarationen

Die Struktur von XML-Dokumenten

Beispiel SYN-9: Einfachstes XML-Dokument

```
<?xml version="1.0"?>
<ausgabe>Hallo Seminarteilnehmer!</ausgabe>
```

Das einfachste mögliche XML-Dokument besteht aus einem **Prolog** und genau einem Element, dem sogenannten **Wurzelement (root element)**. Mehrere Wurzelemente sind nicht erlaubt.

XML-Dokument

Dokument = Prolog + Wurzelement

Prolog

Der Prolog beginnt mit einer sogenannten **XML-Deklaration**, die dem XML-Prozessor anzeigt, dass es sich im folgenden um ein XML-Dokument handelt. Strenggenommen ist sie optional, es gehört jedoch zum guten Stil, sie anzugeben. Darüber hinaus werden im Prolog eventuell dem Dokument zugeordnete **Stylesheets** und **DTDs** referenziert. Der Prolog enthält also Hinweise für die XML-Anwendung, welche weiteren Dateien sie benötigt, um den XML-Inhalt darzustellen.

Beispiel SYN-10: XML-Dokument mit Referenz auf Stylesheet und DTD im Prolog

```
<?xml version="1.0"?>
<?xml-stylesheet href="katalog.xsl" type="text/xsl"?>
<!DOCTYPE katalog SYSTEM "katalog.dtd">

<katalog>
  ....
</katalog>
```

Wurzelement

Das Wurzelement enthält alle anderen Elemente, die immer korrekt hierarchisch (baumartig) verschachtelt sein müssen.

Beispiel SYN-11: Wurzelement

```
<?xml version="1.0"?>

<wurzelement>
  <kindelement1>
    Hallo Seminarteilnehmer
  </kindelement1>
  <kindelement2>
    <enkelement>Schöner Tag heute!</enkelement>
  </kindelement2>
</wurzelement>
```

Beachten Sie:

In der XML-Empfehlung heißt das Wurzelement **Dokument-Element (document element)** oder einfach nur **Wurzel (root)**. Diese Bezeichnung hat sich allerdings im allgemeinen Sprachgebrauch nicht durchgesetzt.

Logische Struktur eines XML-Dokuments

Folgende Tabelle zeigt alle logischen Komponenten, die ein XML-Dokument ausmachen:

Prolog	XML-Deklaration
	Stylesheet-Referenz
	Dokumenttyp-Deklaration
Wurzelement	Elemente (mit Attributen)
	Zeichendaten
	Zeichenreferenzen
	Entity-Referenzen
	CDATA-Abschnitte
	Kommentare
	Verarbeitungsanweisungen
Verschiedenes (Epilog)	Kommentare
	Verarbeitungsanweisungen

Abbildung SYN-4: Logische Struktur eines XML-Dokuments

Physikalische Struktur eines XML-Dokuments

Physikalisch setzt sich ein XML-Dokument aus Daten - im XML-Jargon **Entities** genannt - zusammen.

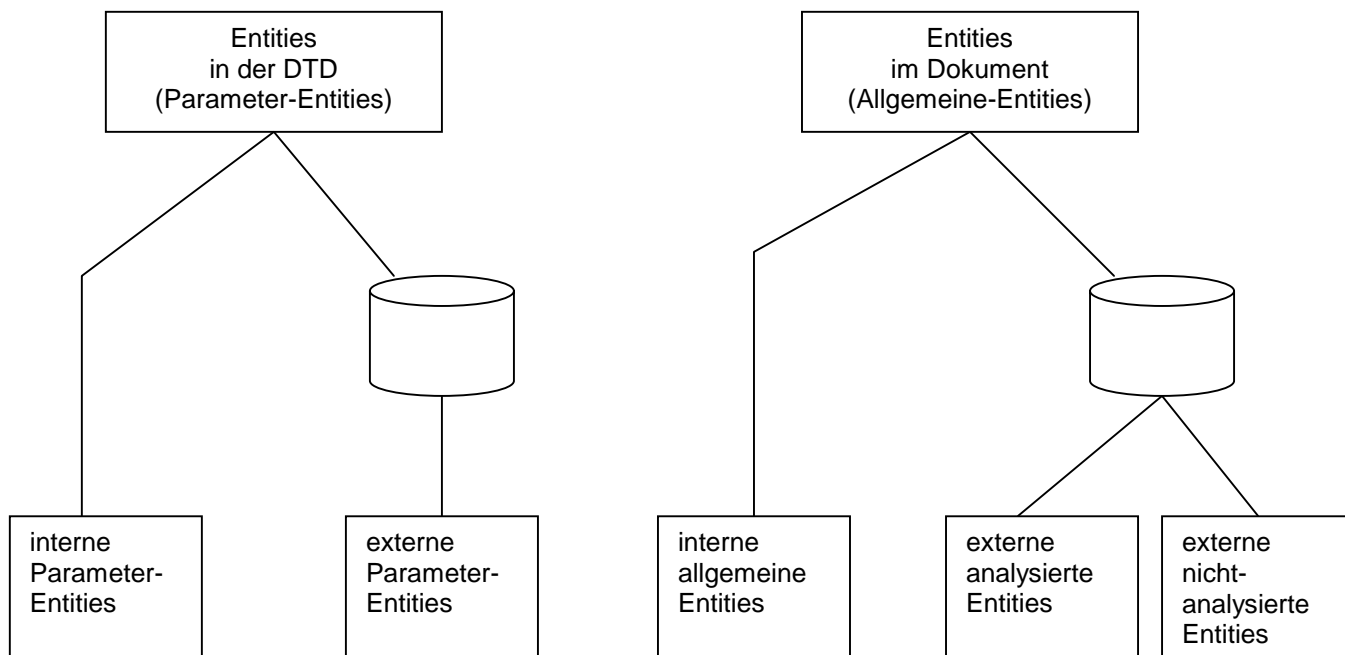


Abbildung SYN-5: Arten von Entities

Das wichtigste Entity ist das **Dokument-Entity (document entity)**, das XML-Dokument selbst.

Interne und externe Entities

Entities können auf andere Entities verweisen und dadurch andere Datenobjekte in das XML-Dokument aufnehmen. So können Sie z.B. Teile anderer XML-Dokumente einbinden (**externes Entity, external entity**). Ein **internes Entity (internal entity)** wird in der Regel in der DTD deklariert. Die Zeichenreferenzen sind ein Sonderfall von internen Entities. Der Inhalt interner Entities wird auch als **Ersetzungstext (replacement text)** bezeichnet, da der Parser an die Stelle im Dokument, an der die Entity-Referenz steht, während der Verarbeitung den referenzierten Text einfügt.

Analysierte und nicht-analysierte Entities

Entities können jegliches Format haben. Handelt es sich um XML-Code, werden sie vom XML-Prozessor verarbeitet (**analysierte Entities, *parsed entities***), sind es z.B. Grafikdateien, werden sie nicht geparkt (**nicht-analysierte Entities, *unparsed entities***), sondern an die Anwendung, die für die Verarbeitung zuständig ist, weitergereicht. Damit der XML-Prozessor weiß, um welche Daten es sich handelt, wird für jedes nicht-analysierte Entity eine Notation in der DTD deklariert. Diese gibt das Format des Entity an z.B., dass es sich um eine Grafik im GIF-Format handelt.

Allgemeine Entities und Parameter-Entities

Schließlich gilt es, **allgemeine Entities (*general entities*)** von **Parameter-Entities (*parameter entities*)** zu unterscheiden. Allgemeine Entities werden im XML-Dokument referenziert, wohingegen Parameter-Entities in der DTD referenziert werden.

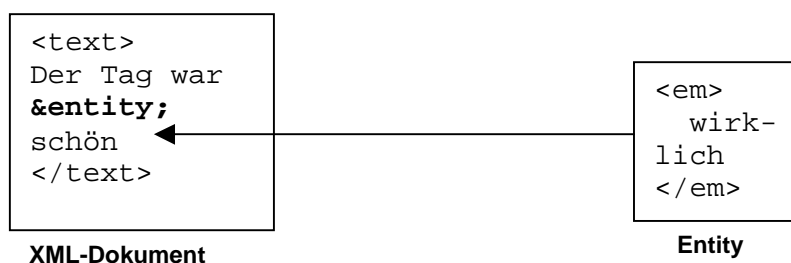
Verwendungszwecke von Entities

- *Wiederverwendung* mehrfach vorkommender Zeichenfolgen (Abkürzungen)
=> interne Entities
- Einbinden systemabhängiger Nicht-XML-Daten in ein XML-Dokument
=> nicht-analysierte Entities
- *Modularisierung* einer DTD
=> Parameter-Entities
- Modularisierung von XML-Dokumenten
=> externe, analysierte Entities

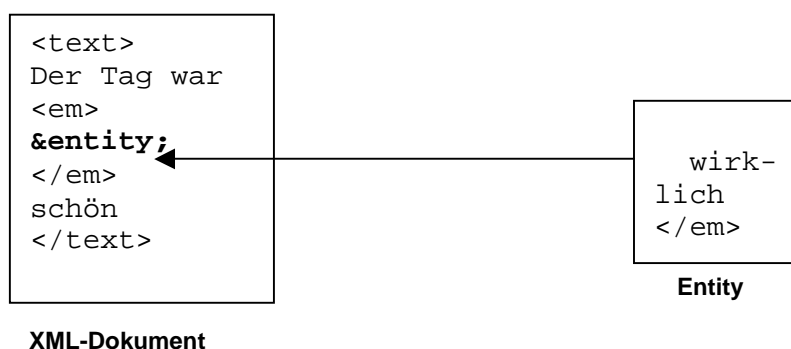
Synchronisation von logischer und physikalischer Struktur

Die logische und physikalische Struktur von XML-Dokumenten müssen miteinander synchronisieren. D.h. Elemente dürfen nicht über die Grenzen eines physischen Entity hinausgehen, und die physischen Entities müssen vollständig in die logischen Elemente eingeschlossen (verschachtelt) sein. Dies soll verhindern, dass z.B. ein eingebundenes externes Entity in einem geöffneten Element referenziert wird, das ein End-Tag zu dem gerade geöffneten Element enthält. Damit wäre die gesamte logische Struktur zerstört. Dieselbe Einschränkung gilt aus Gründen der Einfachheit auch für interne Entities.

Beispiel SYN-12: Synchroner Struktur



Beispiel SYN-13: Asynchrone Struktur



Regeln für die Erstellung von XML-Dokumenten

XML-Deklaration

Typischerweise beginnen XML-Dokumente mit der XML-Deklaration. Eine XML-Deklaration muss mit der Zeichenfolge `<?xml` beginnen und mit der Zeichenfolge `?>` enden.

```
<?xml version="VersionsNr"  
      [encoding="Zeichensatzkodierung"]  
      [standalone="yes|no"]  
?>
```

Versionsinformation

Das `version`-Attribut gibt die XML-Version an. Z.Zt. also immer 1.0.

Beachten Sie:

Streng genommen handelt es sich nicht um ein Attribut. Attribute können nur im Start-Tag eines Elements auftauchen.

Kodierungsdeklaration

Das Attribut `encoding` bestimmt die für das Dokument benutzte Zeichensatzkodierung. Es ist ein optionales Attribut. Wird es nicht angegeben, so erwartet der XML-Prozessor ein Unicode-Dokument in UTF-8 oder UTF-16-Kodierung.

Beachten Sie:

Strenggenommen handelt es sich nicht um ein Attribut, sondern um eine sogenannte **Kodierungsdeklaration (*encoding declaration*)**.

Speichern Sie ihre XML-Dateien direkt in Unicode (z.B. mit Notepad unter Windows 2000), besteht in der Regel keine Notwendigkeit, die Kodierungsdeklaration anzugeben. Speichern Sie ihre XML-Dateien als erweitertes ASCII oder ANSI (*ANSI = American National Standards Institute*) - wie unter Windows üblich -, dann müssen sie als Kodierung den *ISO-Latin-1-Zeichensatz* angeben, um den XML-Prozessor dazu zu bewegen, deutsche Sonderzeichen anzuzeigen.

Beispiel SYN-14: Umlaute in XML-Dateien verwenden

```
<?xml version="1.0" encoding="iso-8859-1"?>
<autor>Günter Grass</autor>
```

Standalone-Dokumentdeklaration

Mit dem Attribut `standalone` können Sie angeben, ob eine DTD zur Analyse des Dokuments geladen werden soll. Wird es nicht gesetzt, so ist die Standardeinstellung `no`. Es ist ein optionales Attribut.

Beachten Sie:

Strenggenommen handelt es sich nicht um ein Attribut, sondern um eine sogenannte **Standalone-Dokumentdeklaration** (*standalone document declaration*).

Dokumenttyp-Deklaration

Die Dokumenttyp-Deklaration ermöglicht die Verknüpfung einer DTD mit einem XML-Dokument. Ein Beispiel soll hier genügen. Im Kapitel DTD-Syntax werden wir uns genauer mit der Einbindung von DTDs in XML-Dokumente beschäftigen.

Beispiel SYN-15: Dokumenttyp-Deklarationen

```
<!DOCTYPE katalog SYSTEM "katalog.dtd">
```

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"DTD/xhtml11-strict.dtd">
```

XML-Namen

- Ein XML-Name besteht mindestens aus einem Zeichen.
- XML-Namen müssen mit einem Buchstaben, einem Unterstrich (`_`) oder einem Doppelpunkt (`:`) beginnen.
- Groß- und Kleinschreibung wird bei XML-Namen beachtet (`<name>`, `<Name>` und `<NAME>` sind drei unterschiedliche Elemente).
- XML-Namen dürfen nicht mit der Zeichenkette `XML` bzw. `xml` beginnen.
- XML-Namen dürfen keine Markup-Symbole enthalten.
- XML-Namen dürfen keine ASCII-Steuerzeichen enthalten (auch keinen Leerraum).
- Alle Buchstaben und Ziffern des Unicode-Zeichensatzes sind erlaubt bis auf einige Ausnahmen, die Sie in der Regel, aber auch nicht in Namen verwenden werden.

weitere nicht erlaubte Unicode-Zeichen in Namen (Auswahl):

Code (hex)	Code (dez)	Zeichen
21	33	!
23	35	#
24	36	\$
25	37	%
29	41	(
	47	/
5B	91	[
5C	92	\
5D	93]
5E	94	^
60	96	~
7B	123	{
7C	124	

Code (hex)	Code (dez)	Zeichen
7D	125	}
7E	126	~

Beachten Sie:

In der Praxis sollte der Doppelpunkt, außer für den Einsatz von Namensraumkennzeichnungen, in XML-Namen nicht genutzt werden.

Es gibt keine Regel, die besagt, dass XML-Namen sinnvoll sein müssen. Sie können Ihren XML-Elementen also jeden beliebigen Namen verleihen. Bedenken Sie jedoch, dass es einer der wichtigsten Vorteile von XML ist, dass der Code selbstbeschreibend ist. Wenn Sie Elemente wie `<dingsda>`, `<irgendwas>` oder `<huh>` verwenden, machen Sie diesen Effekt zunichte. Versuchen Sie, Namen zu verwenden, die wenigstens ansatzweise die Art oder die Aufgabe des Objekts treffen.

Um das Prinzip der Trennung von Darstellung und Inhalt nicht zu durchbrechen, sollten Sie auch Darstellungs-Markup vermeiden!

Beispiel SYN-16: Verschiedene Elementnamen

<code><_produkt></code>	erlaubt
<code><mein produkt></code>	nicht erlaubt
<code><produkt@shop></code>	nicht erlaubt
<code><täter></code>	erlaubt
<code><xmlBuch></code>	nicht erlaubt
<code><дело></code>	erlaubt
<code><test/auto/motor></code>	nicht erlaubt

Elementregeln

- Elemente haben entweder Start- und End-Tag oder sind leer.
- Elemente können **Zeichendaten**, andere Elemente (**Markup**) oder eine Kombination von beiden enthalten (**gemischter Inhalt = *mixed content***).

Beispiel SYN-17: gemischter Inhalt

```
<abschnitt>
  Bestimmte Wörter dieses Absatzes möchte der Autor e-
  ventuell besonders <hervorhe-
  bung>hervorheben<hervorhebung>, um sie zu betonen
</abschnitt>.
```

- Elementnamen unterliegen den Beschränkungen, die für alle XML-Namen gelten.

Attributregeln

- Attribute bestehen immer aus einem Namen und einem Wert
- Der Attribut-Wert steht immer in Anführungszeichen (einzeln oder doppelt)
- Attribut-Namen unterliegen den Beschränkungen, die für alle XML-Namen gelten
- Kein Attribut-Name darf mehr als einmal im selben Start-Tag auftauchen
- Attributwerte dürfen kein < enthalten
- Attributwerte dürfen keine Referenz auf ein externes Entity enthalten
- *Reservierte Attribute:*
lang, space, link, attributes

Regeln für Leerraum

XML kennt folgende Leerraumzeichen (*whitespace*):

Code (hex)	Code (dez)	Beschreibung
09	09	horizontaler Tabulator (<i>Horizontal Tab</i>) (HT)
0A	10	Zeilenvorschub (<i>Line-feed</i>) (LF)
0D	13	Wagenrücklauf (<i>Carriage-return</i>) (CR)

Code (hex)	Code (dez)	Beschreibung
20	32	Leerzeichen (<i>Space character</i>)

Regeln für Leerraum in XML:

- Alle Zeichendaten - also auch die Leerraumzeichen (außer CR!) - werden von einem XML-Prozessor unverändert an die verarbeitende Anwendung weitergereicht. (Beachten Sie den Unterschied zu HTML!)
- Ein *validierender* XML-Prozessor muss darüber hinaus die Anwendung informieren, welche der Leerraumzeichen im Inhalt eines Elements stehen.

Behandlung des Zeilenendes

XML-Dokumente bestehen aus reinem Text, der in Dateien gespeichert wird. Typischerweise enthalten diese Dateien Textzeilen.

Es gibt drei verschiedene Kombinationen von Leerraumzeichen, die benutzt werden, um auf Computersystemen ein Zeilenende darzustellen:

- CR-LF (MS-Windows, MS-DOS)
- LF (Unix, GNU/Linux)
- CR (MacOS)

Ein XML-Prozessor konvertiert alle diese drei Kombinationen zur Kennzeichnung eines Zeilenendes in einen einfachen Zeilenvorschub (Line-Feed, abgekürzt: LF).

Beachten Sie:

Das EBCDIC-Zeichen (*EBCDIC = Extended Binary Coded Decimal Interchange Code*) für einen Zeilenanfang (NEL für engl. *next line*; Zeichenreferenz: #x0085;) wird von XML-Prozessoren nicht korrekt als ein Zeilenendezeichen interpretiert. Dies ist v.a. für die XML-Nutzung auf IBM-Mainframes ein Problem, da die eingesetzten Betriebssysteme OS/390 und MVS den EBCDIC-Zeichensatz verwenden. Hier hat die XML-Empfehlung in der Version 1.0 eine Lücke. Dieser offene Punkt ist ein Thema in den Diskussionen um die neue XML-Version 1.1.

Normalisierung von Attributwerten

Ein XML-Prozessor muss Attributwerte nach bestimmten Regeln *normalisieren*, bevor er sie an die Anwendung weiterreicht bzw. das XML-Dokument validiert. Normalisierung bedeutet, die Anwendung "sieht" den Wert eines Attributs immer in standardisierter (d.h. normalisierter) Form.

Die Normalisierung von Attributwerten ist eines der wenigen "Features" von XML, das die Verarbeitung von XML-Dokumenten kompliziert macht und oftmals Verwirrung stiftet, da validierende und nicht-validierende Parser unter Umständen unterschiedliche normalisierte Attributwerte produzieren.

Die Regeln, denen die Normalisierung von Attributwerten folgt, werden im Modul *DTD* besprochen.

Kommentare

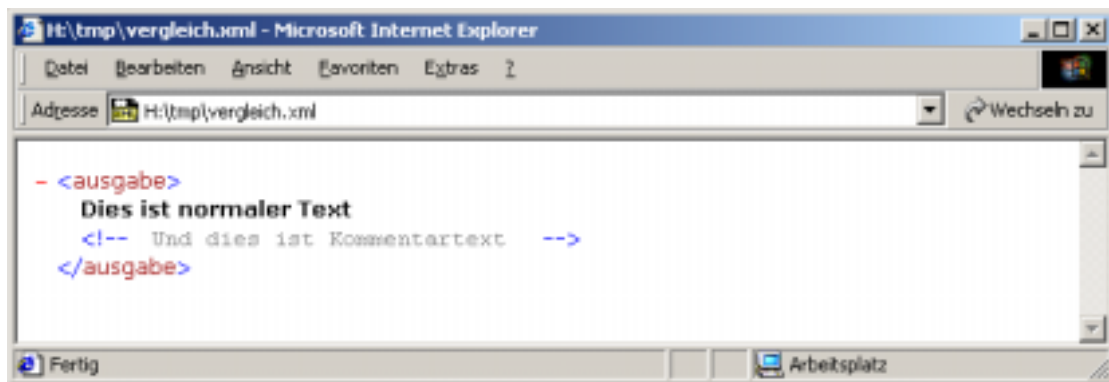
Kommentare sehen in HTML und XML identisch aus

```
<!-- Kommentartext enthält Zeichendaten -->
```

Beispiel SYN-18: Kommentare in XML-Dokumenten

```
<?xml version="1.0"?>
<ausgabe>
  Dies sind Zeichendaten
  <!-- Und dies ist ein Kommentar -->
</ausgabe>
```

Ausgabe:



Ein Kommentar kann an jeder beliebigen Stelle innerhalb eines XML-Dokuments stehen, jedoch nicht innerhalb von anderem Markup.

Beispiel SYN-19: Nicht erlaubter Kommentar

```
<test <!-- nicht erlaubter Kommentar-->>Text</test>
```

Eine weitere nicht erlaubte Kommentarform (doppelter Bindestrich im Kommentarinhalt) zeigt folgendes Beispiel:

Beispiel SYN-20: Doppelte Bindestriche in Kommentaren

```
<!-- Das geht so nicht -- warum? -->
```

Der XML-Prozessor hält die beiden aufeinanderfolgenden Bindestriche (*ein* Bindestrich ist erlaubt!) für die Ankündigung des Kommentarendes und beschwert sich, dass keine schließende spitze Klammer folgt.

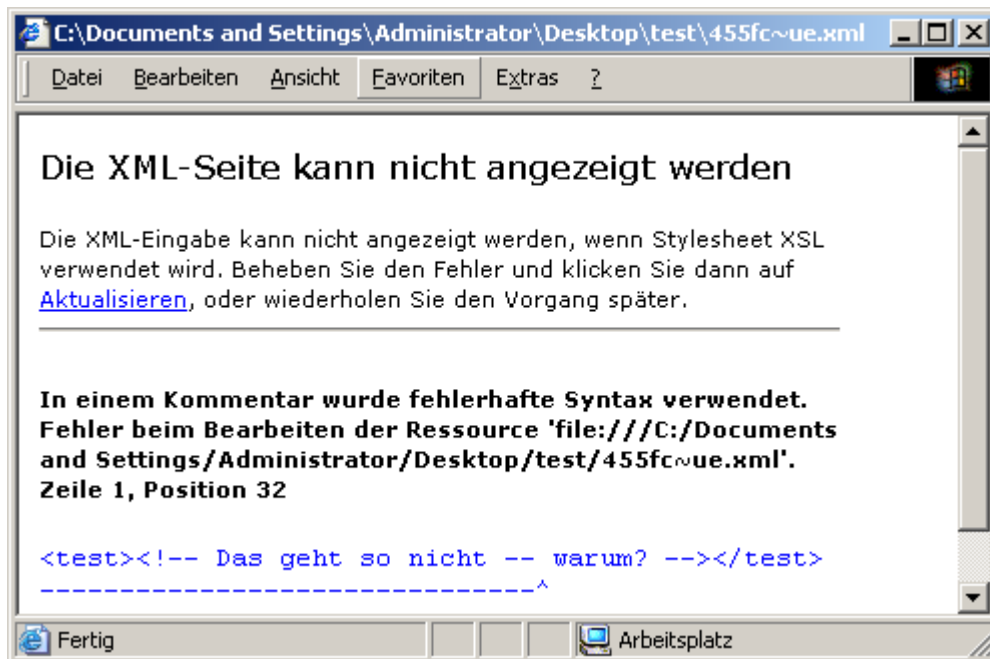


Abbildung SYN-6: Fehlermeldung wegen unerlaubter Zeichen in Kommentar

Beachten Sie:

Sie sollten keine Anweisungen für die Anwendung innerhalb von Kommentaren platzieren, da ein XML-Prozessor nicht verpflichtet ist den Kommentartext an die Anwendung durchzureichen.

Verarbeitungsanweisungen

Verarbeitungsanweisungen (*Processing Instructions*) erlauben es, Befehle (Anweisungen) für andere Programme zur Weiterverarbeitung in einem XML-Dokument unterzubringen. Verarbeitungsanweisungen durchbrechen streng genommen das Prinzip der Trennung von Inhalt und Programmlogik und sollten sparsam eingesetzt werden.

Die allgemeine Syntax für Verarbeitungsanweisungen lautet:

```
<?Ziel Anweisung ?>
```

Ziel ist hier der Bezeichner für die Anwendung, an die die Verarbeitungsanweisung gerichtet ist.

Beispiel SYN-21: Verarbeitungsanweisungen

```
<?xml version="1.0"?>

<?cocoon-process type="xslt"?>

<?xml-stylesheet href="notiz.xslt" type="text/xsl"?>

<notiz>
  <titel>Termine</titel>
  <autor>Knecht Ruprecht</autor>
  <datum>06.12.2001</datum>
  <kommentar>Rute nicht vergessen!</kommentar>
</notiz>
```

Verarbeitungsanweisungen gehören nicht zu den Zeichendaten eines XML-Dokuments, müssen aber in jedem Falle vom XML-Prozessor an die verarbeitende Anwendung weitergereicht werden.

Verarbeitungsanweisungen sind ein umstrittenes XML-Feature, können aber nützlich für folgende **Anwendungsfälle** sein:

- Einbindung von Skripten
- Erweiterung von Dokumenten, ohne Einfluss auf die Gültigkeit des Dokuments
- Transportmittel für Formatierungsinformationen (wie z.B. Seiten- oder Zeilenumbrüche)

Beachten Sie:

Obwohl die XML-Deklaration von der syntaktischen Form her an eine Verarbeitungsanweisung erinnert, ist sie keine solche.

Entity-Referenzen

Vordefinierte Entities und ihre Referenzierung

Zeichen	Zeichenreferenz (hex)	Zeichenreferenz (dez)	Entity-Referenz
&	&	&	&
<	<	<	<

>	>	>	>
'	'	'	'
"	"	"	"

Referenzen auf vordefinierte Entities werden als Substitute für die grundlegenden Markup-Symbole verwendet, damit der XML-Prozessor sie nicht mit Markup-Symbolen verwechselt.

Beispiel SYN-22: Vordefinierte Entity-Referenz <

```
<vergleich>45 &lt; 67</vergleich>
```

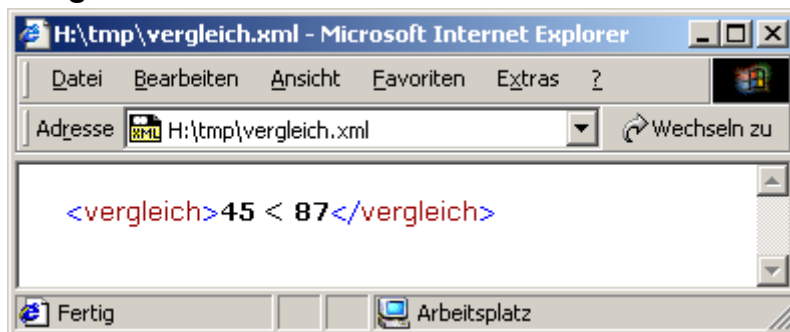
Sie können natürlich auch die numerische Zeichenreferenz nutzen:

Beispiel SYN-23: Numerische Zeichenreferenz für <

XML-Code

```
<vergleich>45 &#60; 67</vergleich>
```

Ausgabe:



Beispiel SYN-24 Fehlender Gebrauch einer Entity-Referenz

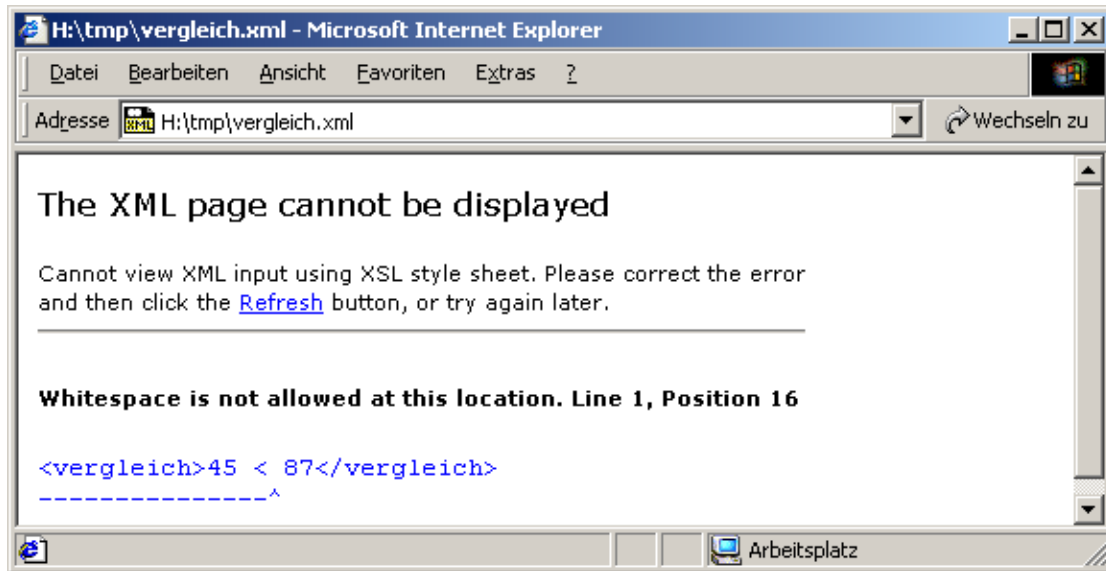
```
<!-- Das Folgende ist kein wohlgeformtes XML! -->
<vergleich>45 < 67</vergleich>
```

```

<!--
    Der XML-Prozessor denkt, es beginnt ein
    neues Tag
-->

```

Ausgabe:



Am einfachsten ist es, die Markup-Symbole immer zu schützen. Strengegenommen gelten jedoch folgende Regeln für den Schutz von Markup-Symbolen:

Entity-Referenz	Gebrauch
&	wird immer genutzt, um das kaufmännische Und (&) zu schützen (außer in CDATA-Abschnitten)
<	wird immer genutzt, um das Kleiner-Als-Zeichen (<) zu schützen (außer in CDATA-Abschnitten)

> ;	<p>Kann genutzt werden, um das Größer-Als-Zeichen (>) zu schützen.</p> <p>Muss genutzt werden, um ein Größer-Als-Zeichen in CDATA-Abschnitten zu schützen, wenn es vor zwei eckigen geschlossenen Klammern steht (]]).</p> <p>Muss genutzt werden, wenn ihr XML-Dokument SGML-kompatibel bleiben soll.</p>
&apos ;	<p>Kann genutzt werden, um ein einfaches Anführungszeichen (') in Literalen zu schützen.</p>
" ;	<p>Kann genutzt werden, um ein doppeltes Anführungszeichen (") in Literalen zu schützen.</p>

Selbstdefinierte Entities

werden in der DTD definiert (s. DTD-Syntax).

Regeln für die Referenzierung von Entities

werden genauer bei der DTD-Syntax betrachtet

CDATA-Abschnitte

Innerhalb eines XML-Dokuments lassen sich bestimmte Abschnitte kennzeichnen, die vom XML-Prozessor nicht verarbeitet werden sollen. Der dort enthaltene Text wird dann nicht interpretiert, sondern "wörtlich" übernommen. Solche CDATA-Abschnitte beginnen mit der Zeichenfolge <![CDATA[und enden mit den Zeichen]]> .

```
<![CDATA[Text, der vom XML-Prozessor ignoriert wird]]>
```

Mit CDATA-Abschnitten lässt sich z.B. die Ersetzung von Entity-Referenzen verhindern oder auch Markup ausgeben.

Beispiel SYN-25: CDATA-Abschnitte

CDATA-Abschnitt

```
<!-- Folgende Schreibweisen liefern identischen Output -
-->

<![CDATA[<beispiel>XML-Beispiele innerhalb eines XML-
Dokuments k&ouml;nnte man gut in einen CDATA-Abschnitt
einbetten</beispiel>]]>
```

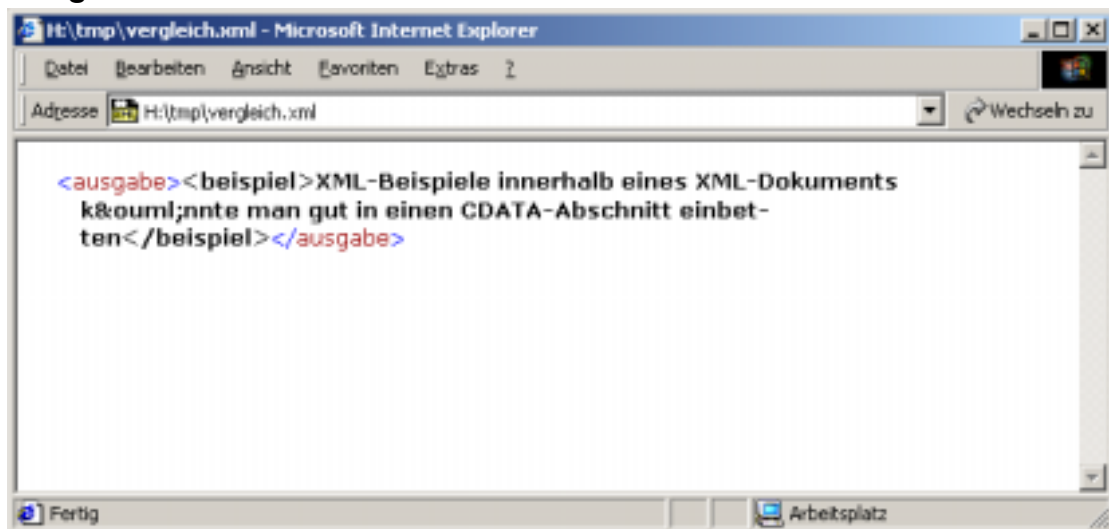
Vordefinierte Entity-Referenzen

```
&lt;beispiel&gt;XML-Beispiele innerhalb eines XML-
Dokuments k&amp;ouml;nnte man gut in einen CDATA-
Abschnitt einbetten&lt;/beispiel&gt;
```

Ausgabe-Code

```
<beispiel>XML-Beispiele innerhalb eines XML-Dokuments
k&ouml;nnte man gut in einen CDATA-Abschnitt einbet-
ten</beispiel>
```

Ausgabe im Browser:



Ausblick: XML 1.1

Zur Zeit arbeitet das W3C an einer neuen Version des XML Standards. Die Änderungen betreffen im wesentlichen die erlaubten Zeichen in XML-Namen und Zeichendaten.

XML 1.1 liegt zur Zeit als Candidate Recommendation vor.

URL:

<http://www.w3.org/TR/xml11/>

Änderungen zu XML 1.0

XML 1.1 beinhaltet Änderungen hinsichtlich

- der erlaubten Zeichen in Zeichendaten (Zeichensatz)
- der erlaubten Zeichen in XML-Namen
- des Prologs
- der erlaubten Zeilenendezeichen

Neu hinzu kommt darüber hinaus eine Regelung hinsichtlich der Unicode-Normalisierung

Änderungen am Zeichensatz

XML 1.0 erlaubt die Verwendung fast aller Unicode-Zeichen zugelassen. Ausnahme sind die Surrogat-Blöcke und die Zeichen `#xFFFFE` und `#xFFFFF` sowie die C0-Controls mit Ausnahme der Leerraumzeichen *Horizontaler Tabulator* (`#x09`), *Zeilenvorschub* (`#x0A`) und *Wagenrücklauf* (`#x0D`).

XML ändert dies folgendermaßen:

- Die in XML 1.0 noch erlaubte direkte Verwendung der sogenannten C1-Controls (Zeichenbereich `#x80` bis `#x9F`) und des `DEL`-Zeichens (für engl. *delete*; `#x7F`) wird auf die Verwendung als Zeichenreferenz eingeschränkt.

Ausnahme:

Das in der IBM-Mainframe-Welt übliche Newline-Zeichen `NEL` (`#x85`) ist explizit erlaubt.

- Die sogenannten C0-Controls (#x00 bis #x1F) werden nun als Zeichenreferenzen in Zeichendaten zugelassen.

Ausnahme:

Das NUL-Zeichen (#x00) bleibt weiterhin verboten und die schon in XML 1.0 als Leerraum zulässigen Zeichen *Horizontaler Tabulator* (#x09), *Zeilenvorschub* (#x0A) und *Wagenrücklauf* (#x0D) sowie das in der IBM-Mainframe-Welt übliche Newline-Zeichen NEL (#x85) sind weiterhin auch direkt verwendbar.

Diese Änderungen führen dazu, dass (bestimmte) XML-1.1-Dokumente keine wohlgeformten XML-1.0-Dokumente mehr sind. XML 1.1 ist somit nicht vollständig rückwärtskompatibel. Dies ist auch der Grund für den Versionssprung.

Änderungen bei XML-Namen

- Definition von zulässigen Zeichen in XML-Namen folgt nun dem Paradigma: "Alles was nicht verboten ist, ist erlaubt". In XML 1.0 ist die Definition rigide: "Alles was nicht erlaubt ist, ist verboten"; und damit weniger flexibel hinsichtlich den Anpassungen an neue Unicode-Standards.
- Es sind praktisch alle Zeichen erlaubt, außer solchen, die Trennzeichen darstellen bzw. als solche Verwendung finden. Weiterhin bleiben Ziffern und Verbindungszeichen (*Combining Characters*) für den Namensanfang ausgeschlossen.

Diese Änderung lässt sich am handgreiflichsten an den veränderten Produktionsregeln für XML-Namen nachvollziehen. Falls Sie dieses Thema interessiert, sollten Sie direkt in der Spezifikation nachschlagen und die Produktionsregeln [4] und [5] aus der XML-1.0-Empfehlung mit den geänderten Versionen der XML-1.1-Empfehlung vergleichen.

Änderungen am Prolog

- Die Versionsinformation der XML-Deklaration lässt nun den Wert 1.1 zu, damit ein Parser die Versionen unterscheiden kann. Dies ist besonders wichtig vor dem Hintergrund, dass XML-1.0-Dokumente und XML-1.1-Dokumente nicht vollständig kompatibel sind.
- Ein XML-1.1-Prozessor sollte allerdings (muss aber nicht!) auch XML-1.0-Dokumente akzeptieren

Änderungen der Zeilenendekventionen

XML 1.0 interpretiert lediglich die Zeichen Wagenrücklauf (*Carriage Return*, CR) und Zeilenvorschub (*Line Feed*, LF) bzw. die Sequenz CR-LF als Zeilenendezeichen.

XML 1.1 fügt folgende Zeichen(-kombinationen) hinzu:

- eine CR-NEL-Sequenz (#x0D,#x85)
- ein einzelnes NEL-Zeichen (#x85)
- ein einzelnes Unicode-Zeilentrennzeichen (#x2028)

Unicode-Normalisierung

Neben der Kategorie der Wohlgeformtheit und Gültigkeit wird nun die Eigenschaft *normalisiert* für XML-Dokumente eingeführt. Mit dem Unicode-Zeichensatz können nämlich viele Zeichen unterschiedlich dargestellt werden. So kann z.B. ein kleines ä mit dem einzelnen Zeichen U+00E4 (LATIN SMALL LETTER A WITH DIAERESIS) oder als Kombination der beiden Zeichen U+0061 (LATIN SMALL LETTER A) und U+0308 (COMBINING DIAERESIS) dargestellt werden. Dies ist in einigen Fällen von Vorteil, erschwert jedoch den binären Vergleich zweier Zeichenketten. Daher definiert das W3C in einem eigenen Zeichenmodell (*Character Model*) eine Normalisierung, bei der alle Zeichenkombinationen zu einem einheitlichen definierten Zeichen konvertiert werden.

Detailliertere Informationen zum Zeichenmodell des W3C finden Sie im Working Draft *Character Model for the World Wide Web 1.0*

URL:

<http://www.w3.org/TR/charmod/>

Die XML-1.1-Empfehlung verlangt, dass ein XML-1.1-Prozessor auf Normalisierung prüfen sollte (nicht muss!).

Ein Algorithmus zur Prüfung auf Normalisierung ist allerdings sehr komplex und in der Entwicklung entsprechend aufwendig. Z.Zt. gibt es daher noch keinen Parser, der dieses Feature unterstützt.

Beachten Sie:

Ein Parser normalisiert nicht, sondern prüft lediglich auf Normalisierung.

Relevanz und Einsatz von XML 1.1

Der etwas polemische, aber für den Regelfall in der Praxis wohl beste Ratschlag zum Umgang mit XML 1.1 lautet:

Benutzen Sie es nicht.

Tatsächlich werden Sie XML 1.1 nur dann verwenden müssen, wenn Sie Mongolisch, Kambodschanisch oder einige andere Sprachen außerhalb des europäischen Raumes sprechen und ihre XML-Dokumente in diesen Sprachen erstellen müssen.

Ansonsten führt die nicht gegebene Rückwärtskompatibilität von XML 1.1 eher zu neuen Problemen als dass sie welche löst.

Implementierungen von XML 1.1

Name	Lizenz	Entwickler	Sprachen	ab Version
Xerces-J	open source	Apache XML Project	Java	2.3.0
URL: http://xml.apache.org/xerces2-j/index.html				
Xerces-C++	open source	Apache XML Project	C++	2.3.0
URL: http://xml.apache.org/xerces-c/index.html				
RXP	open source	Richard Tobin	C	1.4.0 PRE 4
URL: http://www.cogsci.ed.ac.uk/~richard/rxp.html				

Beachten Sie:

Keine der Implementierungen unterstützt Unicode-Normalisierung.