

# **XML Converter**

## **Handbuch und Referenz**

Michael Seume

Lizenzbestimmungen:

Copyright © 2008-2020 Michael Seume

Die Software und die Dokumentation sind urheberrechtlich geschützt.

Die Software und die Dokumentation werden vom Autor zur Verfügung gestellt.

Hiermit wird unentgeltlich jeder Person, die eine Kopie der Software und der Dokumentationen erhält, die Erlaubnis erteilt, diese uneingeschränkt zu benutzen.

Die Weitergabe ist unter den gleichen Bedingungen möglich. Die Software (xmlcnv) und die Lizenzbedingungen (lizenz.txt) müssen dazu unverändert weitergegeben werden.

Die Software und die Dokumentation werden ohne jede ausdrückliche oder implizierte Garantie bereitgestellt, einschließlich der Garantie zur Benutzung für den vorgesehenen oder einen bestimmten Zweck. Für Fehler in der Software und Schäden, die sich aus der Nutzung der Software ergeben, wird keine Haftung übernommen.

Marken:

Linux is a registered trademark of Linus Torvalds.

macOS is a trademark of Apple Inc., registered in the U.S. and other countries.

OS X is a trademark of Apple Inc., registered in the U.S. and other countries.

Unicode is a registered trademark of Unicode, Inc. in the United States and other countries.

Windows is a registered trademark of Microsoft Corporation in the United States and other countries.

Impressum:



XML Converter (xmlcnv)

Version 5.0.2 vom 27.09.2020

© 2008–2020 Michael Seume,

Michael Seume, Dachwig, Deutschland

Alle Rechte vorbehalten!

Alle genannten Marken, Produkt- und Firmennamen  
sind Eigentum ihrer Urheber.

[info@xml-converter.de](mailto:info@xml-converter.de)



# Neues

*Version Datum - Beschreibung*

5.0.2 27.09.2020

Änderungen:

- Der No-Spy-Code für die Windowsversion (eingesetzt seit XML Converter 4.0.5 vom 12.10.2016) wurde wieder entfernt.
- Optimierung des Programmcodes für Windows und macOS.

Updates:

- UltraEdit 27.00 vom 19.05.2020
- PSPad 5.0.3 vom 09.02.2020
- Notepad++ 7.8.6 vom 21.04.2020
- Tincta 4.0.1 vom 23.02.2020
- TextPad 8.4.0 vom 17.04.2020

5.0.1 08.02.2020

Änderungen:

- UTF-8-Parser (mit positionsgenauer Fehleranzeige)
- UTF-16-Parser (mit positionsgenauer Fehleranzeige)
- UTF-16LE-Parser (mit positionsgenauer Fehleranzeige)
- UTF-16BE-Parser (mit positionsgenauer Fehleranzeige)
- UTF-32-Parser (mit positionsgenauer Fehleranzeige)
- UTF-32LE-Parser (mit positionsgenauer Fehleranzeige)
- UTF-32BE-Parser (mit positionsgenauer Fehleranzeige)
- Korrektur der UTF-Encoder.

- 5 02.02.2020  
Änderungen:
- Optimierung der Skript-Verarbeitungsgeschwindigkeit. Je nach Daten und Skript ist eine Beschleunigung bis zu 200x gegenüber der Vorversion möglich.
  - Verbessertes Speichermanagement (Hinweis: Extrem große Dateien sollten mit der 64-Bit-Version verarbeitet werden. Der verfügbare freie Hauptspeicher sollte je nach Struktur der Daten das 8- bis 16-fache der Daten betragen).
  - Korrektur eines Kodierungsfehlers der Vorversion.
- 4.1.0 08.12.2019  
Neues:
- Tracking über den Kommandozeileschalter `-track` oder `trackautostart` (siehe Bedienung).
- 4.0.17 13.10.2019  
Änderungen:
- Code Signatur für macOS Catalina.
- 4.0.16 16.06.2019  
Änderungen:
- Problembeseitigung bei der Verarbeitung negierter regulärer Ausdrücke bei Beachtung der Groß-/ Kleinschreibung.
- 4.0.15 13.05.2019  
Änderungen:
- Im XML-Parser wurden die Zeichenbereiche für Bezeichner (Elementnamen, Attributnamen) auf die Bereiche `xC0-xD6` und `xD8-xF6` und `xF8-xFF` erweitert, womit z.B. deutsche Umlaute hier zulässig sind.
- Updates:
- UltraEdit 26.00 vom 27.02.2019
  - UE Studio 19.00 vom 29.03.2019
  - PSPad 5.0.1 vom 22.02.2019
  - Notepad++ 7.6.6 vom 04.04.2019

- 4.0.14 01.01.2019  
Änderungen:
- Die 32 Bit-Version für macOS wurde endgültig eingestellt. Für macOS ist ab jetzt ausschließlich die 64 Bit-Version verfügbar.
- Korrekturen:
- Probleme beim Sortieren von Eingabedateien behoben.
  - Probleme bei der Gesamtausgabe in eine Datei in eine andere Ordnerstruktur behoben.
- Updates:
- PsPad 5.0.0 vom 24.4.2018
  - NotePad++ 7.5.8 vom 23.7.2018
- 4.0.13 19.08.2018  
Erweiterungen:
- Der Parser übergeht im Modus WARNINGS=SMART jetzt auch fehlerhafte Attributwerte. So erfolgt bei den reservierten Zeichen (<), (>) und (&) in diesem Modus kein Abbruch. Es wird jedoch jeweils eine Warnung ausgegeben.
- 4.0.12 1.5.2018  
Erweiterungen:
- Erweiterter Unicodebereich außerhalb der Unicode BMP 0x10000 ... 0x10FFFF implementiert.
- Korrekturen:
- Probleme beim Vergleich unterschiedlicher Maßeinheiten beseitigt (Zoll vs. Zentimeter).
- 4.0.11 17.12.2017  
Korrekturen:
- Fehler beim IMPLIED-Kontextabfragen innerhalb von Toplevel-Elementen behoben.
- Updates
- NotePad++ 7.5.3 vom 5.12.2017
  - UltraEdit 24.20 vom 31.8.2017

Weitere Einträge siehe Entwicklungsgeschichte am Ende des Dokumentes (siehe Entwicklungsgeschichte).





# Inhaltsverzeichnis

<b>Einleitung .....</b>	<b>5</b>
<b>Konventionen.....</b>	<b>8</b>
<b>Bedienung .....</b>	<b>9</b>
<b>Programmierung.....</b>	<b>14</b>
Prinzipielle Arbeitsweise .....	14
So funktioniert ein Skript .....	16
Wir erstellen ein Skript .....	19
Stapelverarbeitungsdatei erstellen.....	21
Optimierung des Skriptes.....	22
<b>Sprachbeschreibung .....</b>	<b>23</b>
Anweisung .....	24
Attributbedingung.....	25
Bedingung .....	29
Bedingungsausdruck .....	30
Element-Ausgabeanweisung .....	32
Element-Handler .....	33
Element-Konvertierungsregel .....	34
Entitäten-Ausgabeanweisung .....	35
Entitäten-Handler .....	36
Entitäten-Konvertierungsregel .....	37
Kommentar .....	38
Kontextbedingung .....	39
Regulärer Ausdruck .....	41
Skript.....	44
Skript-Einstellungen .....	45
Unterelement .....	46
Unterstruktur .....	47
Verzweigung .....	48
Zeichenkette .....	50
Zeichenkette, ein ge schränk te .....	51
<b>Sprachreferenz.....</b>	<b>52</b>
" " .....	52
' ' .....	54
[ ].....	56
//.....	61
/* */ .....	62
AFTER .....	63
AFTERX.....	66
ALTERNATE .....	69
ASCII.....	71
ATTLIST .....	73
BASE .....	74



BEFORE .....	76
BEFOREX .....	79
BEG .....	82
CHR .....	86
CONVERTDATA .....	87
CONVERTELEMENTS .....	88
CONVERTENTITIES .....	90
COPY .....	91
COUNT .....	95
CUT .....	98
DAT .....	101
DEL .....	104
ELEMENT .....	106
ELSE .....	108
ELSEIF .....	109
EMPTY .....	110
END .....	112
ENDE .....	116
ENDIF .....	117
ENTITY .....	118
ERR .....	122
FILE .....	124
FIND .....	126
FIRST .....	129
IF .....	131
IMPLIED .....	132
IN .....	138
INX .....	142
LAST .....	145
MODEL .....	147
NAMECASE .....	148
NOT .....	149
NOTAFTER .....	152
NOTAFTERX .....	153
NOTBEFORE .....	154
NOTBEFOREX .....	155
NOTEMPTY .....	156
NOTFIND .....	157
NOTFIRST .....	158
NOTIN .....	159
NOTINX .....	160
NOTLAST .....	161
NOTSUB .....	162
NOTTOPLEVEL .....	163
PUBLIC .....	164
READERENCODING .....	165
SUB .....	166
SYNTAX .....	168



SYSTEM .....	170
TOKEN.....	172
TOPLEVEL.....	176
VALUE .....	177
WARNINGS .....	184
WRITERENCODING .....	185
X.....	186
<b>Fehlermeldungen.....</b>	<b>187</b>
<b>XML/SGML-Glossar .....</b>	<b>188</b>
ANY.....	189
ATTLIST .....	190
Attribut.....	192
Attributname.....	193
Attributwert.....	194
Ausführungsanweisung.....	195
Ausschluss.....	196
Bereiche, bedingte.....	197
Bereiche, markierte.....	198
Bezeichner.....	200
BOM.....	202
CDATA.....	203
DOCTYPE.....	205
Dokument.....	207
DTD.....	208
Einschluss.....	209
ELEMENT.....	210
EMPTY.....	212
Element.....	214
Elementname.....	215
ENTITY.....	216
Entität.....	219
Kommentar.....	220
Entitätenname.....	221
Instanz.....	222
Leerraum.....	223
NDATA.....	224
NAMECASE.....	225
NOTATION.....	226
OMITTAG.....	227
PCDATA.....	228
Prolog.....	229
PUBLIC.....	230
RCDATA.....	231
SDATA.....	232
SGML-Deklaration.....	233
SHORTTAG.....	235
SUBDOC.....	237



---

SYSTEM .....	238
UCS-2 .....	239
UCS-4 .....	240
UTF-8 .....	241
UTF-16 .....	242
UTF-32 .....	243
Vereinbarung .....	244
Vereinbarung, er wei ter te .....	245
Vereinbarung, lee re .....	246
XML-Deklaration .....	248
Zeichenverweis .....	250
<b>Editoren .....</b>	<b>251</b>
Editoren mit integrierter Unterstützung für XML-Converter .....	251
Konfigurationsanleitungen .....	253
<b>Systemanforderungen .....</b>	<b>268</b>
Wohlgeformtes SGML .....	272
Wohlgeformtes XML .....	275
<b>Entwicklungsgeschichte .....</b>	<b>277</b>
<b>Quellen .....</b>	<b>307</b>
Literatur .....	307
Editoren .....	307

# Einleitung

Der XML Converter ist

- eine **Skriptsprache** zur XML-Konvertierung, zur SGML-Konvertierung und zur HTML-Konvertierung (HTML5),
- ein **Interpreter** (Konsolenprogramm), dessen Steuerung über Kommandozeilenparameter erfolgt und für dessen Ausführung ein Konvertierungsskript benötigt wird, welches für den Anwendungsfall **programmiert** werden muss,
- für **Linux**, **macOS** und **Windows** geeignet.

Um den XML-Converter nutzen zu können, sollten Sie

- über Kenntnisse in SGML, XML und/oder HTML (HTML5) und
- Programmiererfahrungen in Skriptsprachen

besitzen.

Weiterhin benötigen Sie einen guten Text-Editor (siehe Kapitel Editoren).

Los geht es ...

Der XML Converter besteht aus einem Parser und einer integrierten Skriptsprache.

Der Parser liest die Daten als sequentiellen Datenstrom. Die Daten werden dabei ereignisorientiert ausgewertet. Während des Lesens werden beim Auftreten von Start- und Endtags der Elemente oder beim Auftreten von Entitäten Ereignisse ausgelöst und die betreffenden Rückruffunktionen im Skript (*Element-Handler*, *Entitäten-Handler*) aufgerufen.

Die Ereignisse werden also parallel zum Einlesen ausgeführt.

Nichtbehandelte Ereignisse und nichtbehandelte Datenbereiche werden unverändert in den Ausgabedatenstrom ausgegeben. Bestandteile der Daten, die Ereignisse auslösen, werden selbst nicht ausgegeben. Für deren Ausgabe ist die Rückruffunktion im Skript verantwortlich.

Beispieldokument:

```
<?xml version="1.0" ?>
<buch>
  <kapitel>
    <titel nr="1">Einf&#252;hrung</titel>
  </kapitel>
</buch>
```

Beim Parsen des Dokumentes werden folgende Ereignisse in der angegebenen Reihenfolge ausgelöst:

```
ELEMENT buch (BEG)
ELEMENT kapitel (BEG)
ELEMENT titel (BEG)
```



```
ENTITY #252
ELEMENT titel (END)
ELEMENT kapitel (END)
ELEMENT buch (END)
```

Der Parser ruft bei jedem Ereignis die entsprechende Rückruffunktion (*Element-Handler*, *Entitäten-Handler*) auf. Wenn keine Rückruffunktion definiert ist, werden die ereignisauslösenden Daten unverändert ausgegeben.

Das Beispielskript:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<script>

ELEMENT kapitel
BEG=""
END=""
ENDE

</script>
```

entfernt das Element `kapitel` komplett:

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

  <titel nr="1">Einf&#252;hrung</titel>

</buch>
```

Mit dem folgenden Beispiel:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<script>

ELEMENT buch
BEG="<html>"
END="</html>"
ENDE

ELEMENT kapitel
BEG=""
END=""
ENDE
```



```
ELEMENT titel
ATTLIST [nr=""]
BEG='<h1>'+VALUE([nr])+". "
END="</h1>"
ENDE
```

```
ENTITY #252="&uuml;"
</script>
```

**wird ein komplettes HTML-Dokument erzeugt:**

```
<html>
<h1>1. Einf&uuml;hrung</h1>
</html>
```



# Konventionen

Dieses Handbuch ist als Online-Handbuch konzipiert. Sie sollten es als PDF-Dokument mit einem geeigneten Betrachter (Adobe Reader o.ä.) verwenden. Alle relevanten Begriffe sind zur besseren Nutzbarkeit im Text durch Hyperlinks gekennzeichnet. In der gedruckten Form finden Sie die verwiesenen Begriffe in der Kopfzeile jeder Seite ähnlich einem Lexikon.

Bitte beachten Sie folgenden Konventionen des Handbuches.

**Name** Bezeichnungen in fett (innerhalb der Syntax): Schlüsselworte, Operatoren und Syntaxzeichen der Skriptsprache.

*Name* Bezeichnungen in kursiv (innerhalb der Syntax): Syntaxeinheit. Diese Begriffe werden in den Legenden der jeweiligen Syntaxbeschreibung erläutert.

{ } Syntaxgruppe. Diese wird verwendet, um das Auftreten der Bestandteile zu definieren. Das Auftreten wird durch einen anhängenden Operator qualifiziert:

{...} (ohne Operator), genau ein Auftreten  
{...}? optional, kein Auftreten oder ein Auftreten,  
Beispiel: {Anweisung}?  
{...}\* optional, kein, ein oder mehrere Auftreten,  
Beispiel: {Anweisung}\*  
{...}+ ein oder mehrere Auftreten.  
Beispiel: {Anweisung}+

Bitte beachten Sie, dass die Syntaxklammern "{" und "}" und die dazugehörigen Operatoren "?", "\*" und "+" lediglich zur Sprachbeschreibung verwendet werden, aber in der Sprache selbst nicht vorkommen.

Die Syntaxklammer ist vergleichbar der SGML- und XML-Syntax für die Beschreibung der Inhaltsmodelle. In SGML- und XML wird aber die normale Klammer "(...)" verwendet.

Syntaxeinheiten und Schlüsselworte können mit folgenden Operatoren verbunden werden:

| logisches **oder**  
{*Element1*|*Element2*}  
Element 1 oder Element 2

,

definierte Reihenfolge:  
{*Element1*,*Element2*}  
Element2 muss dem Element1 folgen

## Bedienung

Der XML-Converter ist ein Konsolenprogramm. Die Steuerung erfolgt über Kommandozeilenparametern. Zur Ausführung benötigen Sie ein Konvertierungsskript (siehe *Skript*), welches Sie für Ihren Anwendungsfall programmieren müssen (siehe *Wir erstellen ein Skript*).

Der typische Anwendungsfall ist die Konvertierung von XML-Daten in ein Medienformat, wie z.B. HTML. Innerhalb des Skriptes definieren Sie auf einfachste Weise die Umsetzung der Elemente und Entitäten gemäß Ihrer DTD in die Auszeichnungsform des Zielformates.

### Syntax

```
xmlcnv eing skript ausg -s -w -q -o -l -track
```

**eing** Eingabedateiname oder Dateispezifikation der zu konvertierenden Datei oder Dateien. Es dürfen Platzhalterzeichen ("\*" und "?") in der Dateispezifikation eingegeben werden. Bei der Eingabe von Platzhalterzeichen werden alle Dateien, die dem Suchmuster entsprechen, konvertiert.

Beispiele:

Vollständige Dateibenennung

```
"C:\Test\Demo.xml"  
"/Users/Mac/Test/Demo.xml"  
"/home/linux/Test/Demo.xml"
```

oder mit Wildcards

```
"C:\Test\*.xml"  
"/Users/Mac/Test/*.xml"  
"/home/linux/Test/*.xml"
```

oder

```
"C:\Test\TE?T.xml"  
"/Users/Mac/Test/Te?t.xml"  
"/home/linux/Test/Te?t.xml"
```

oder relative Pfadangabe

```
"Test/*.xml"  
"Test/Te?t.xml"
```

oder indirekte, relative Pfadangabe

```
".\..\Test\*.xml".  
"..../Test/*.xml".
```



- skript** Projektname. Es genügt der Skriptname ohne Erweiterung (z.B. "Html"). Die Projektdatei sollte sich im gleichen Verzeichnis wie das Programm befinden. Die Standardextension ist ".xb";  
Beispiel:
- ```
"C:\Test\Html"  
"/Users/Mac/Html"  
"/home/linux/Html"
```
- oder
- ```
"C:\Test\Html.xb"  
"/Users/Mac/Html.xb"  
"/home/linux/Html.xb"
```
- oder, wenn sich der Converter im gleichen Verzeichnis befindet:
- ```
"Html"
```
- oder
- ```
"Html.xb"
```
- ausg** Ausgabedateiname oder Dateierweiterung der Ergebnisdateien. Bei Angabe nur einer Dateierweiterung, wird für jede Eingabedatei eine Ausgabedatei mit dieser Erweiterung erzeugt. Bei Angabe eines Dateinamens, wird eine Ausgabedatei erstellt, unabhängig davon, ob eine einzelne Eingabedatei angegeben wurde oder eine Dateispezifikation zur Bearbeitung mehrerer Eingabedateien. Bei Angabe einer Dateierweiterung wird für jede Eingabedatei eine Ausgabedatei im gleichen Ordner erzeugt. Bei Angabe einer Ausgabedatei wird diese, wenn kein Ordner angegeben wurde, im aktuellen Arbeitsverzeichnis ausgegeben. Bei Weglassen der Angabe *ausg* werden die Eingabedateien nur geprüft.  
Beispiele:  
Vollständige Dateibenennung des Ergebnisses:

```
"C:\Test\Seite.htm"  
"/Users/Mac/Seite.htm"  
"/home/linux/Seite.htm"
```

oder Erweiterung

```
"htm"
```

**-s** schaltet die Dateisuche in Unterverzeichnissen ein  
**-w** aktiviert die zusätzliche Ausgabe von Fehlermeldungen direkt in den Ausgabedaten. Dieser Schalter ist für Webanwendungen zu empfehlen. Die Fehlermeldungen werden dann im Browser (Ausgabedaten HTML) angezeigt.  
**-q** minimale Konsolenausgabe (bei Erfolg keine Konsolenausgabe)

- o schaltet die Sortierung der gefunden Dateien ein
  - l Spracheinstellung für die Sortierung:
    - g deutsch (Syntax: -lg)
    - e englisch (Syntax: -le)
    - f französisch (Syntax: -lf)
  - track Aktiviert die Ausgabe einer HTML-Trackingdatei. Der Dateiname der Trackingdatei wird aus dem Dateinamen der Zieldatei gebildet, wobei der Name mit dem Zusatz `_track` erweitert wird. Die Dateiendung ist `html`. Beispiel: Die Zieldatei hat den Namen `test.htm`, die Trackingdatei heißt dann `test_track.html`. Die Trackingdatei muss mit einem Browser angezeigt werden.
  - trackautostart Wie `-track` wird eine HTML-Trackingdatei erzeugt, wobei die Wiedergabe beim Aufruf der Trackingdatei automatisch startet.
- Der Erfolg der Programmausführung wird durch den Rückgabewert (Errorlevel) angezeigt (siehe Fehlermeldungen).

### Hinweis

Mehrere Kommandozeilenparameter müssen jeweils mit einem Leerzeichen voneinander abgetrennt werden!

### Hinweis

Dateinamen oder Ordernamen, die Leerzeichen enthalten, müssen komplett in Anführungszeichen gesetzt werden.

### Hinweis Linux und Mac

Bei der Verwendung von Platzhalterzeichen müssen die Parameter komplett in Anführungszeichen gesetzt werden.

### Beispiel 1 Windows

```
xmllcnv C:\Test\Doku.xml Html C:\Test\DOKU.htm
```

Konvertiere das Dokument `Doku.xml` mit dem Skript `Html`. Erstelle eine Ausgabedatei `Doku.Htm`.

### Beispiel 1 macOS

```
xmllcnv /Users/Mac/Doku.xml Html /Users/Mac/DOKU.htm
```

Konvertiere das Dokument `Doku.xml` mit dem Skript `Html`. Erstelle eine Ausgabedatei `Doku.Htm`.

### Beispiel 1 Linux

```
xmllcnv /home/linux/Doku.xml Html /home/linux/DOKU.htm
```



Konvertiere das Dokument `Doku.xml` mit dem Skript `Html`. Erstelle eine Ausgabedatei `Doku.Htm`.

### Beispiel 2 Windows

```
xmllcncv C:\Test\*.xml Html HTM -s
```

Konvertiere alle XML-Dateien im Verzeichnis `C:\Test` und allen Unterverzeichnissen mit dem Skript `Html.xb`. Erstelle für jede gefundene XML-Datei eine Ausgabedatei mit der Dateierweiterung `htm`.

### Beispiel 2 macOS

```
xmllcncv /Users/Mac/test/*.xml Html HTM -s
```

Konvertiere alle XML-Dateien im Verzeichnis `/Users/Mac/test` (`Mac` = Benutzername) und allen Unterverzeichnissen mit dem Skript `Html.xb`. Erstelle für jede gefundene XML-Datei eine Ausgabedatei mit der Dateierweiterung `htm`.

### Beispiel 2 Linux

```
xmllcncv /home/linux/test/*.xml Html HTM -s
```

Konvertiere alle XML-Dateien im Verzeichnis `/home/linux/test` (`linux` = Benutzername) und allen Unterverzeichnissen mit dem Skript `Html.xb`. Erstelle für jede gefundene XML-Datei eine Ausgabedatei mit der Dateierweiterung `htm`.

### Beispiel 3 Windows

```
xmllcncv Test\*.xml Html Test\Geamt.htm -s -o -lg
```

Konvertiere alle XML-Dateien im Unterverzeichnis `\Test` und allen Unterverzeichnissen mit dem Skript `Html.xb`, wobei die Dateien in geordneter Reihenfolge unter Berücksichtigung deutscher Umlaute abgearbeitet werden. Erzeugt eine einzige Ausgabedatei mit dem Namen `Gesamt.htm`.

### Beispiel 3 macOS

```
xmllcncv Test/*.xml Html Test/Geamt.htm -s -o -lg
```

Konvertiere alle XML-Dateien im Unterverzeichnis `\Test` und allen Unterverzeichnissen mit dem Skript `Html.xb`, wobei die Dateien in geordneter Reihenfolge unter Berücksichtigung deutscher Umlaute abgearbeitet werden. Erzeugt eine einzige Ausgabedatei mit dem Namen `Gesamt.htm`.

### Beispiel 3 Linux

```
xmllcncv Test/*.xml Html Test/Geamt.htm -s -o -lg
```

Konvertiere alle XML-Dateien im Unterverzeichnis `/Test` und allen Unterverzeichnissen mit dem Skript `Html.xb`, wobei die Dateien in geordneter



Reihenfolge unter Berücksichtigung deutscher Umlaute abgearbeitet werden. Erzeugt eine einzige Ausgabedatei mit dem Namen `Gesamt.htm`.

# Programmierung

## Prinzipielle Arbeitsweise

---

Der Converter geht bei der Konvertierung wie folgt vor:

### 1. Laden des Skriptes

Einlesen und überprüfen des Skriptes.

### 2. Einlesen der Daten (Instanzen)

Für jede Datei wird ein Konvertierungsprozess gestartet. Beachten Sie, dass ausschließlich Instanzen von XML und SGML bearbeitet werden können.

### 2. Erstes Parsen

Prüfen der Daten. Dabei wird anhand des skripteigenen Modells eine Prüfung vorgenommen. Standardmäßig werden alle auftretenden Elemente, Attribute und Entitäten auf Übereinstimmung mit der Skriptdefinition geprüft. Eine Prüfung der syntaktischen Anordnung der Elemente gemäß DTD erfolgt nicht!

### 3. Zweites Parsen = Konvertieren

Während des zweiten Parse-Prozesses werden abhängig vom Objekttyp (Element, Entität, Text) folgende Aktionen zur Konvertierung ausgelöst:

Elemente	Bei jedem Auftreten eines Elementes wird eine Aktion ausgelöst, die das betreffende Unterprogramm ( <i>Element-Handler</i> ) im Skript ausführt. In diesen Unterprogrammen werden die Konvertierungsregeln für die Elemente ausgeführt. Das Konvertierungsergebnis wird in Reihenfolge der Programmabarbeitung direkt in die Ausgabedatei geschrieben.
Entitäten	Bei jedem Auftreten einer Entität wird eine Aktion ausgelöst, die das betreffende Unterprogramm ( <i>Entitäten-Handler</i> ) im Skript ausführt. In diesen Unterprogrammen werden die Konvertierungsregeln für die Entitäten ausgeführt. Das Konvertierungsergebnis wird in Reihenfolge der Programmabarbeitung direkt in die Ausgabedatei geschrieben.
Text	Alle Textdaten werden, wenn diese nicht ausdrücklich durch Skriptanweisungen unterdrückt werden sollen, direkt und unverändert in die Ausgabedatei geschrieben.



Beachten Sie, dass die original XML-Informationen wie der Prolog, Element-Start- und Endtags, Entitäten, Ausführungsanweisungen und Kommentare ausgeblendet werden.

Ohne Konvertierungsregeln zur Ausgabe werden nur die textuellen Informationen zur Ausgabedatei übernommen!

## So funktioniert ein Skript

Das Skript basiert intern selbst auf XML. Es durchläuft bei der Prüfung den gleichen Parser wie die Daten.

Auf eine explizite Auszeichnung in der XML-Syntax, also mit Tags wie `<ELEMENT>` wurde aber verzichtet. Das Skript kann aber auch als XML-Dokument erstellt werden:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<script>

</script>
```

Die wichtigsten Befehle sind `BEG` und `END` in der Syntax einer Zuweisungsoperation:

```
BEG="..."
END="..."
```

Es handelt sich hierbei um Zuweisungen an den Datenstrom des Parsers. Der Parser durchläuft alle Elemente, Zeichendaten und Entitäten des Dokumentes und erzeugt einen Datenstrom der enthaltenen Zeichendaten.

Diesem Datenstrom können an den Elementgrenzen, also dem Starttag und dem Endtag, Daten hinzugefügt werden. `BEG` steht für den Datenstrom beim Erreichen des Starttags und `END` für den Datenstrom beim Erreichen des Endtags.

Die Zuweisungsoperationen des *Element-Handler* `autor`:

```
ELEMENT autor
BEG='<p class="autor">'
END="</p>"
ENDE
```

fügt an Stelle des Starttags `<autor>` den Text `<p class="autor">` ein und an Stelle des Endtags den Text `</p>`.

Dieses Skript kann bei identischem Ergebnis auch wie folgt geschrieben werden:

```
ELEMENT autor
BEG="<p "
BEG="class="
BEG=CHR(34)
BEG="autor"
BEG=CHR(34)
BEG=">"
END="</ "
END="p>"
```

ENDE

Da `BEG` und `END` von getrennten Ereignissen angesteuert werden, werden diese nie zeitgleich ausgeführt. Bei leeren Elementen wird nur das Ereignis für den Starttag, also `BEG`, ausgeführt!

Die Ausführung der Anweisungen erfolgt sequentiell jeweils für `BEG` und für `END` getrennt. Das folgende Beispiel erzeugt immer noch das gleiche Ergebnis wie das vorhergehende Beispiel:

```
ELEMENT autor
BEG="<p "
END="</ "
BEG="class="
BEG=CHR(34)
BEG="autor"
BEG=CHR(34)
END="p>"
BEG=">"
ENDE
```

Derartige Zerstückelungen sollten aber zu Gunsten der besseren Lesbarkeit vermieden werden.

Im folgenden Beispiel wird eine *Element-Handler* für das Element `autor` definiert. Für den Beginn (`BEG`) des Elementes wird der Starttag `<p class="autor">` in die Ausgabe eingefügt und für das Ende (`END`) des Elementes der Endtag `</p>`. Der Element-Handler wird mit `ENDE` beendet.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<script>
```

```
ELEMENT autor
BEG='<p class="autor">'
END="</p>"
ENDE
```

```
</script>
```

Da das Skript selbst auf XML basiert, kann es auch so geschrieben werden:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<script>
```

```
ELEMENT
autor
```



```
BEG
=
'<p class="autor">'
END
=
"</p>"
ENDE
```

</script>

**oder so**

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<script>ELEMENT autor BEG='<p class="autor">' END="</p>"
ENDE</script>
```



## Wir erstellen ein Skript

---

Die Programmierung einer Konvertierungsanwendung benötigen Sie einen Texteditor, eine DTD und Beispieldokumente (-Instanzen).

Verwenden Sie für XML-Skripte einen Texteditor mit **UTF-8** Modus. Wenn Sie ein Skript für SGML erstellen, verwenden Sie bitte einen Texteditor mit "normalen Textmodus" (ASCII, US-ASCII bzw. **ANSI**).

### 1. Anlegen der Skript-Datei

Erstellen Sie eine neue Skriptdatei mit einem Texteditor und fügen Sie folgenden Text ein:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<script>

</script>
```

### 2. Elementregeln

Anhand der DTD können Sie das Grundgerüst des Skriptes (siehe auch *Skript*) aufbauen. Für jedes *Element* Ihrer DTD (z.B. `<!ELEMENT kapitel titel ...>`) erstellen Sie ein passendes Elementhandle (siehe *Element-Handler* bzw. ELEMENT):

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<script>

ELEMENT kapitel
ENDE

ELEMENT titel
ENDE

</script>
```

Mit den Konvertierungsregeln wird das Zielformat der Strukturkonvertierung gesteuert. Die grundlegenden Anweisungen dazu sind die BEG- und END-Anweisung.

Die BEG-Anweisung gibt alle angegebenen Daten am Beginn des betreffenden Elementes aus, wobei bei mehreren Anweisungen die Ausgabe in Reihenfolge der Abarbeitung erfolgt. Die END-Anweisung steuert Ausgaben am Ende des jeweiligen Elementes.

Ein einfaches Beispiel ist dafür die Umsetzung von Hervorhebungen:

```
ELEMENT fett
BEG='<b>'
END='</b>'
```

ENDE

Durch Kontextabfragen lassen sich sehr einfach die betreffende Formatierungen ermitteln. So werden im Beispiel die Überschriften (`titel`) in der ersten Ebene (`kapitel`) mit `h1` gekennzeichnet, die in der zweiten Ebene mit `h2` und alle anderen mit `h3`:

```
ELEMENT titel
IF IN kapitel IN kapitel IN kapitel
  BEG="<h3>"
  END="</h3>"
ELSEIF IN kapitel IN kapitel
  BEG="<h2>"
  END="</h2>"
ELSE
  BEG="<h1>"
  END="</h1>"
ENDIF
ENDE
```

### 3. Entitätenregeln

Am Ende des Skriptes fügen Sie die *Entitäten-Konvertierungsregel* hinzu, welche jeweils für jede in der DTD definierte Entität die passende Zielform definiert (Für das Zielformat HTML können viele Entitäten 1:1 übernommen werden. Andere Zielformate erfordern aber ein gezieltes Umcodieren der Entitäten!):

```
ENTITY auml="&auml;"
```

Siehe *Skript*.



## Stapelverarbeitungsdatei erstellen

---

### Linux und macOS:

Erstellen Sie eine Textdatei:

```
xmlcnv test.xml xmlhtml.xb test.html
```

Speichern Sie die Datei unter dem Namen:

```
Test.sh
```

Öffnen Sie das Terminal und machen Sie die Datei ausführbar:

```
chmod +x test.sh
```

Zum Ausführen starten Sie die Stapelverarbeitungsdatei im Terminal:

```
./test.sh
```

### Windows:

Erstellen Sie eine Textdatei:

```
xmlcnv test.xml xmlhtml.xb test.html
```

Speichern Sie die Datei unter dem Namen:

```
Test.bat
```

Starten Sie die Stapelverarbeitungsdatei durch Aufruf in einem Konsolenfenster (Start/cmd.exe):

```
Test.bat
```

oder direkt per Doppelklick aus dem Windows-Explorer.

## Optimierung des Skriptes

---

Bei der Entwicklung des Skriptes werden Sie Elementregeln für die vorkommenden Elemente nacheinander entwickeln. Das Skript wird in der Standardeinstellung Elemente, die definiert worden sind, konvertieren und Elemente, die nicht definiert worden sind, unverändert ausgeben. Für eine schnelle Skriptentwicklung ist dieses Verfahren gut geeignet.

Für die **Qualitätssicherung** des Skriptes ist diese Methode aber ungeeignet. Das Problem besteht darin, dass eine Konvertierung nur dann vollständig ist, wenn alle vorkommenden Elemente und Entitäten für eine Konvertierung definiert worden sind.

Wenn Sie Ihr Skript in der ersten Entwicklungsphase fertig gestellt haben, fügen Sie dem Skript am Beginn folgende Schalter hinzu:

```
CONVERTELEMENTS=ALL  
CONVERTENTITIES=ALL
```

Mit diesen Einstellungen wird festgelegt, dass für alle in den Daten vorkommenden Elemente und Entitäten auch entsprechende Regeln im Skript definiert sein müssen.

Das Skript erzeugt dann beim Auftreten eines nichtdefinierten Elementes oder einer nichtdefinierten Entität eine Fehlermeldung und bricht die Konvertierung ab.

Bei Abfragen nach Attributwerten werden typischerweise Bedingungen in der Form [colsep="0"] oder [colsep<>"1"] definiert. Bei den Abfragen "=" und "<>" werden die Abfragen aber ohne Beachtung von Groß-/Kleinschreibung ausgeführt und zusätzlich bei numerischen Inhalten nach dem numerischen Wert. Die Ausführung dieser Vergleichsoperationen benötigt natürlich zusätzliche Zeit. Benötigen Sie die Nichtbeachtung der Groß-/Kleinschreibung oder den numerischen Vergleich nicht, können die Ausführungszeit optimieren, wenn Sie die Abfragen == und != stattdessen verwenden. Diese Abfragen vergleichen die Werte nach exaktem Inhalt, die enthaltenen Zeichen müssen also gleich sein.

### Hinweis

Diese Einstellung ist für die Konvertierung von Daten dringend zu empfehlen!

### Siehe

*Skript-Einstellungen.*



## Sprachbeschreibung

Übersicht aller Sprachbestandteile in alphabetischer Reihenfolge.

Aufbau des *Skriptes* und seiner Sprachbestandteile:

- *Skript-Einstellungen*
- *Element-Handler*
- *Entitäten-Handler*
- *Verzweigungen*
- *Anweisungen*
- *Kommentare*

### Hinweis

Folgen Sie einfach den Verweisen in der Beschreibung *Skript*.

### Hinweis

Beachten Sie, dass alle Anweisungen separat in der Sprachreferenz beschrieben werden.

***Anweisung***

Eine Anweisung ist eine syntaktische Programmeinheit, welche eine Aktion im Programm (*Skript*) darstellt.

- *Element-Ausgabeeanweisung*
- *Entitäten-Ausgabeeanweisung*



## **Attributbedingung**

---

Eine *Bedingung*, die den Wert eines *Attributs* abfragt.

Für die Abfrage eines vererbten Attributwertes muss IMPLIED verwendet werden.

### **Syntax 1**

**[Attributname Operator Wert]**

*Attributname*

*Bezeichner* für den Namen eines *Attributes*.

*Operator*

Vergleichsoperator:

<	das Attribut ist kleiner als der Wert
>	das Attribut ist größer als der Wert
<=	das Attribut ist kleiner gleich dem Wert
>=	das Attribut ist größer gleich dem Wert
<>	das Attribut ist ungleich dem Wert (unabhängig der Groß- und Kleinschreibung)
=	das Attribut ist gleich dem Wert (unabhängig der Groß- und Kleinschreibung)
==	das Attribut ist identisch dem Wert (mit Beachtung der Groß- und Kleinschreibung)
==^	das Attribut beginnt mit dem Wert (unabhängig der Groß- und Kleinschreibung)
==^	das Attribut beginnt mit dem Wert (mit Beachtung der Groß- und Kleinschreibung)
==\$	das Attribut endet mit dem Wert (unabhängig der Groß- und Kleinschreibung)
==\$	das Attribut endet mit dem Wert (mit Beachtung der Groß- und Kleinschreibung)
==*	das Attribut enthält den Wert (unabhängig der Groß- und Kleinschreibung)
==*	das Attribut enthält den Wert (mit Beachtung der Groß- und Kleinschreibung)



<b>=~</b>	das Attribut trifft den angegebenen regulären Ausdruck (unabhängig der Groß- und Kleinschreibung)
<b>==~</b>	das Attribut trifft den angegebenen regulären Ausdruck (unter Beachtung der Groß- und Kleinschreibung)
<b>&lt;</b>	das Attribut ist nicht kleiner als der Wert
<b>&gt;</b>	das Attribut ist nicht größer als der Wert
<b>&lt;=</b>	das Attribut ist nicht kleiner gleich dem Wert
<b>&gt;=</b>	das Attribut ist nicht größer gleich dem Wert
<b>=</b>	das Attribut ist nicht gleich dem Wert (unabhängig der Groß- und Kleinschreibung)
<b>==</b>	das Attribut ist nicht identisch dem Wert (mit Beachtung der Groß- und Kleinschreibung)
<b>=^</b>	das Attribut beginnt nicht mit dem Wert (unabhängig der Groß- und Kleinschreibung)
<b>==^</b>	das Attribut beginnt nicht mit dem Wert (mit Beachtung der Groß- und Kleinschreibung)
<b>=</b>	das Attribut endet nicht mit dem Wert (unabhängig der Groß- und Kleinschreibung)
<b>==</b>	das Attribut endet nicht mit dem Wert (mit Beachtung der Groß- und Kleinschreibung)
<b>=*</b>	das Attribut enthält nicht den Wert (unabhängig der Groß- und Kleinschreibung)
<b>==*</b>	das Attribut enthält nicht den Wert (mit Beachtung der Groß- und Kleinschreibung)
<b>=~</b>	das Attribut trifft nicht den angegebenen regulären Ausdruck (unabhängig der Groß- und Kleinschreibung)
<b>==~</b>	das Attribut trifft nicht den angegebenen regulären Ausdruck (unter Beachtung der Groß- und Kleinschreibung)

*Wert*

*Zeichenkette, für den Attributwert.*



## Syntax 2

### [Attributname1 Operator Attributname2]

*Attributname1*

*Bezeichner* für den Namen eines *Attributes*. Diese Attribut wird bei Kontextabfragen vom gefundenen Element gelesen.

*Operator*

Vergleichsoperator:

<	kleiner
>	größer
<=	kleiner gleich
>=	größer gleich
<>	ungleich (unabhängig der Groß- und Kleinschreibung)
=	gleich (unabhängig der Groß- und Kleinschreibung)
==	identisch (mit Beachtung der Groß- und Kleinschreibung)
!=	nicht gleich (unabhängig der Groß- und Kleinschreibung)
!==	nicht identisch (mit Beachtung der Groß- und Kleinschreibung)

*Attributname2*

*Bezeichner* für den Namen eines *Attributes*. Dieses Attribut wird bei Kontextabfragen vom aktuellen Element gelesen.

Abfrage des Attributwertes des aktuellen Elementes oder des angegebenen Elementes.

## Beispiel

Daten:

```
<kapitel id="TBB.4">
  <titel>Berechnung</titel>
</kapitel>
```

Skript:

```
ELEMENT kapitel
  ATTLIST [id=""]
  IF [id<>"" ]
    BEG='<a name="" +VALUE([id]) + "' />'
  ENDIF
ENDE
```

Ergebnis:



```
<a name="TBB.4"/>Berechnung
```

Wenn das Attribut `id` nicht leer ist wird es als HTML-Anker `a` ausgegeben.

**Siehe auch**

*Attributname, Attributwert, Zeichenkette, VALUE, IMPLIED*

***Bedingung***

Eine Bedingung ist eine syntaktische Programmeinheit, die im Ergebnis einen Wahrheitswert (`Richtig` oder `Falsch`) zurückliefert.

**Siehe auch**

AFTER, BEFORE, BEG, EMPTY, END, FIND, FIRST, IN, INX, IMPLIED, LAST, SUB, TOPLEVEL

***Bedingungsdruck***

---

Ein Bedingungsdruck ist eine Menge von *Bedingungen*, die im logischen Ergebnis den Wahrheitswert (*Richtig* oder *Falsch*) liefern. Alle Bedingungen werden im Bedingungsdruck mit einer logischen Und-Verknüpfung geprüft, d.h., ein Bedingungsdruck ist nur dann *Richtig*, wenn alle Einzel-*Bedingungen* *Richtig* liefern.

**Syntax 1**

(EMPTY | FIND | SUB | VALUE | IMPLIED | *Attributbedingung* | *Kontextbedingung*)+

**Syntax 2**

{BEG | END}, (EMPTY | FIND | SUB | VALUE | IMPLIED | *Attributbedingung* | *Kontextbedingung*)\*

**Hinweis**

Die Abfragen BEG und END müssen am Beginn der Abfrage positioniert werden. Alle anderen Abfragen lassen sich beliebig kombinieren.



## ***Element-Ausgabeanweisung***

---

Eine Element-Ausgabeanweisung ist eine *Anweisung*, die der Ausgabe von Konvertierungsergebnissen bei der Elementkonvertierung dient. Diese kommen in den *Element-Konvertierungsregel* für die *Elemente* in der Syntax der *Anweisungen* BEG und END vor.

### **Syntax**

{*Zeichenkette* | CHR | COPY | COUNT | CUT | DEL | FILE | IMPLIED |  
TOKEN | VALUE}

*Zeichenkette*                      Ein an dieser Position auszugebender Text.

### **Beispiel**

```
ELEMENT titel
BEG="Text"+CHR(32)+COUNT(kapitel)
END=VALUE(kapitel[id])+COPY(b,ONE,FULL)
ENDE
```

... einfache Ausgabeanweisungen.

### **Siehe auch**

*Zeichenkette*, BEG, CHR, COPY, COUNT, CUT, DEL, END, FILE, IMPLIED, TOKEN, VALUE, *Element-Konvertierungsregel*



## ***Element-Handler***

---

Ein Element-Handler ist ein Ereignisprogramm vom Typ `ELEMENT`. Der Parser ruft bei jedem Auftreten eines Starttags oder Endtags des definierten Elementes dieses Ereignisprogramm auf und führt die enthaltenen Konvertierungsregeln aus.

### **Syntax**

`{ELEMENT}+`

### **Beispiel**

```
ELEMENT fett
ENDE
```

Dieses Beispiel definiert ein Element-Handle für das Element `fett`. Bei jedem Vorkommen des Elementes `b` wird dieses Programm aufgerufen und ausgeführt.

### **Siehe auch**

*Skript*, *Entitäten-Handler*, `ELEMENT`



## ***Element-Konvertierungsregel***

---

Eine Menge von Anweisungen, die zur Konvertierung von Elementen bestimmt sind.

### **Syntax**

{*Verzweigung*|BEG|END|ERR|DAT}+

<i>Verzweigung</i>	Programmverzweigungen (IF-ELSEIF-ELSE-END-IF)
BEG	Ersetzungsregeln für den Starttag
END	Ersetzungsregeln für den Endtag
ERR	Fehlermeldung
DAT	Typ der Datenkonvertierung

### **Siehe auch**

*Skript*, *Element-Handler*, ELEMENT, *Element-Ausgabeeinweisung*, *Zeichenkette*, BEG, CHR, COPY, COUNT, CUT, DEL, END, FILE, IMPLIED, TOKEN, VALUE



### ***Entitäten-Ausgabeanweisung***

---

Eine Entitäten-Ausgabeanweisung ist eine *Anweisung*, die der Ausgabe von Konvertierungsergebnissen bei der Entitätenkonvertierung dient. Diese kommen in den *Entitäten-Konvertierungsregel* vor.

#### **Syntax**

{*Zeichenkette*|VALUE|CHR}

*Zeichenkette*

Ein an dieser Position auszugebender Text.

#### **Hinweis**

Rekursionsschutz! Bei Verwendung von VALUE innerhalb einer *Entitäten-Ausgabeanweisung* werden nur Textbestandteile (CDATA) des betreffenden *Attributwerts* zurückgegeben und wiederum enthaltene Entitäten ausgelassen!



### ***Entitäten-Handler***

---

Ein Entitäten-Handler ist ein Ereignisprogramme vom Typ ENTITY. Der Parser ruft bei jedem Auftreten der definierten Entität dieses Ereignisprogramm auf und führt die enthaltenen Konvertierungsregeln aus.

#### **Syntax**

**{ENTITY {ALTERNATE}? {ASCII}?}**+

ENTITY                      Entitäten-Definition mit Ersetzungsregeln

ALTERNATE                 optionale alternative Ersetzungsregeln.

ASCII                        optionale ASCII-Ersetzungsregeln.

#### **Siehe auch**

*Skript, Element-Handler*, ENTITY, ALTERNATE, ASCII

***Entitäten-Konvertierungsregel***

---

Eine Menge von Anweisungen, die zur Konvertierung von Entitäten bestimmt sind.

**Syntax**

{ALTERNATE ASCII}+



## **Kommentar**

---

### **Beschreibung**

Die Kommentaranweisungen (`//` und `/* */`) sind in allen Whitespace-Bereichen zulässig. Whitespace-Bereiche sind Trennbereiche zwischen Anweisungen, die in der Regel aus Leerzeichen, Tabulatoren und Zeilenwechslern bestehen. Auch innerhalb von Anweisungen sind Whitespace-Bereiche entsprechend der Anweisungs-Syntax zulässig.

### **Syntax 1**

*// Kommentarzeile*

*Kommentar*                      Kommentarzeile, bestehend aus beliebigem Text, der mit einem Zeilenwechsel (Enter, CR LF) abgeschlossen wird.

### **Syntax 2**

*/\* Kommentarbereich \*/*

*/\**                                      Beginn des Kommentarbereiches.

*Kommentarbereich*                beliebiger Textbereich. Dieser Bereich kann sich auch über mehrere Zeilen erstrecken.

*\*/*                                      Ende des Kommentarbereiches.

### **Typ**

*Skript*

### **Siehe auch**

*//, /\* \*/, Skript*

## ***Kontextbedingung***

---

Prüfung der Kontextsituation des aktuellen Elementes innerhalb des Dokumentes.

### **Syntax**

{IN | INX | BEFORE | AFTER | FIRST | LAST | TOPLEVEL}

Erläuterungen (auszugsweise):

- **IN *element***  
Befindet sich das aktuelle Element im unmittelbaren Kontext des angegebenen Elementes? Ist das angegebene Element also das Elternelement des aktuellen Elementes?
- **INX *element***  
Befindet sich das aktuelle Element im erweiterten Kontext des angegebenen Elementes? Ist das angegebene Element also ein beliebiges übergeordnetes Element?
- **AFTER *element***  
Befindet sich das aktuelle Element direkt hinter dem angegebenen Element?
- **BEFORE *element***  
Befindet sich das aktuelle Element direkt vor dem angegebenen Element?
- **FIRST**  
Ist das aktuelle Element das erste Element innerhalb des Elternelementes?
- **LAST**  
Ist das aktuelle Element das letzte Element innerhalb des Elternelementes?
- **TOPLEVEL**  
Ist das aktuelle Element das Basiselement des Dokumentes?

Diese Befehle können mit dem NOT-Präfix kombiniert werden.

Diese Kontextabfragen lassen sich beliebig kombinieren, wobei folgende Regeln zu beachten sind:

- Die einzelnen Bedingungen werden nacheinander geprüft. Die Auswertung wird abgebrochen, wenn eine Bedingung den Wahrheitswert `Falsch` liefert.
- Wird als Parameter einer Abfrage ein Element **ohne Klammern** angegeben und liefert diese Abfrage `Richtig`, dann wird dieses



Element für die folgende Abfrage als das aktuelle Element betrachtet.  
Das Element des Parameters wird also zum aktuellen Element erklärt!

- Wird als Parameter einer Abfrage ein Element **in Klammern** angegeben, bleibt das aktuelle Element unverändert.

### Beispiel 1

```
ELEMENT absatz
IF IN eintrag IN liste INX kapitel
    BEG="Ein Absatz in einer Liste im Kapitel."
ENDIF
ENDE
```

### Beispiel 2

```
ELEMENT absatz
IF INX kapitel LAST
    BEG="Ein Absatz im letzten Kapitel"
ENDIF
ENDE
```

### Hinweis

Die Abfrage der Kontextbedingungen ist der Kernbestandteil des Converters. Mit diesen Abfragen lassen sich strukturierte Daten "in Form" bringen. So lässt sich das erste Element einer Liste mit der Abfrage `FIRST` ermitteln und anders gestalten, als die folgenden Listenelemente. Auch das letzte Element kann mit der Abfrage `LAST` ermittelt und gesondert layoutet werden. Listen in einer Tabelle können mit der Abfrage `INX table` ermittelt und layoutet werden. Die Kontextabfragen sind beliebig kombinierbar.



## Regulärer Ausdruck

---

Für die Prüfung von Attributwerten können reguläre Ausdrücke eingesetzt werden.

.	(Punkt) Beliebiges Zeichen, ausgenommen Zeilenwechsel (CR, LF, FF). Beispiel: "Beispieltext" Der Ausdruck "sp.*?xt" findet "spieltext". Bitte beachten: Wenn im Datenbereich Zeilenwechsel vorhanden sind, müssen diese im Ausdruck berücksichtigt werden, Beispiel: "Beginn(.\n)*?Ende".
^	Beginn der Zeichenkette Der nachfolgende angegebene Ausdruck muss mit dem Beginn der Zeichenkette übereinstimmen. Beispiel: "Beispieltext" Der Ausdruck "^Bei" trifft .
	Logische Oder-Verknüpfung Der Linke und der rechte Ausdruck werden oder- verknüpft.
()	Gruppe Klammerausdrücke werden zur Gruppierung von logischen Einheiten verwendet. Beispiel: (eins zwei drei) trifft die Worte eins, zwei oder drei
\$	Ende der Zeichenkette Der davor angegebene Ausdruck muss mit dem Ende der Zeichenkette übereinstimmen.
\b	Wortgrenze Der vorhergehende Ausdruck trifft Wortzeichen und hinter diesen Zeichen befindet sich ein Leerraum oder das Ende der Zeichenkette. Oder der nachfolgende Ausdruck trifft Wortzeichen und vor diesem Zeichen befindet sich ein Leerraum oder der Beginn der Zeichenkette.



\B	Keine Wortgrenze
\d	Ziffer (0...9)
\D	Keine Ziffer
\n	Zeilenumbruch (0x0a)
\r	Zeilenvorschub (0x0d)
\s	Leerraum trifft Leerzeichen, Tabulator, Zeilenwechsel
\S	Kein Leerraum
\t	Tabulator
\w	Wort trifft ein Wort bestehend aus Buchstaben, Zahlen und dem Unterstrich (A-Z, a-z, 0-9, _).
\W	Kein Wort.
\x	Die nachfolgenden Hexadezimalzeichen werden als Zeichncode interpretiert. \x3E   trifft das >-Zeichen (0x3e)
\	Das nachfolgende Zeichen wird ohne Interpretation als das angegebene Zeichen gesucht. Folgende Zeichen müssen mit vorangestellten umgekehrten Schrägstrich (\) angegeben werden: ^ \$ \ . * + ?   ( ) [ ] { } Beispiel: \^       trifft das ^-Zeichen \\$       trifft das \$-Zeichen \.       Trifft den Punkt
[ <i>Zeichenbereich</i> ]	Ein Zeichen aus dem angegebenen <i>Zeichenbereich</i> [abc] [a-z] [a-z0-9] [0-9\.\+\-]



[ <i>Zeichenbereich</i> ]	Ein Zeichen, welches nicht dem angegebenen <i>Zeichenbereich</i> entspricht [ <sup>^</sup> 0-9]
*	Das Auftreten des vorangehenden Ausdrucks ist: kein, einmal oder mehrmalig
+	Das Auftreten des vorangehenden Ausdrucks ist: einmal oder mehrmalig
?	Das Auftreten des vorangehenden Ausdrucks ist: kein oder einmal.
{ <i>Anz</i> }	Der vorhergehende Ausdruck trifft exakt <i>Anz</i> -Mal. [0-9]{3} trifft exakt 3 Ziffern
{ <i>Min</i> , }	Der vorhergehende Ausdruck trifft mindestens <i>Min</i> -Mal. [0-9]{3,} trifft mindestens drei aufeinanderfolgende Ziffern
{ <i>Min</i> , <i>Max</i> }	Der vorhergehende Ausdruck trifft mindestens <i>Min</i> -Mal und maximal <i>Max</i> -Mal. [0-9]{3,5} trifft Zahlen, die mindestens 3 Ziffern und maximal 5 Ziffern enthalten

### Hinweis

Da reguläre Ausdrücke im Skript innerhalb von Zeichenketten angegeben werden, darf das Begrenzungszeichen der Zeichenkette selbst nicht innerhalb des regulären Ausdrucks verwendet werden.

Verwenden Sie in diesem Fall für das doppelte Anführungszeichen den Code `\x22` und für das einfache Anführungszeichen den Code `\x27`.



## Skript

---

Das Skript enthält alle Anweisungen zur Konvertierung der XML- oder SGML-Daten in ein definiertes Format. Ein Skript muss entsprechend der folgenden Syntax (in der korrekten Reihenfolge) aufgebaut sein.

### Syntax

*Skript-Einstellungen*

*{Element-Handler}?*

*{Entitäten-Handler}?*

### XML, Hinweis zur Kodierung

Wenn UTF-8 Daten verwendet werden sollen, so kann es sinnvoll sein, das Skript selbst als UTF-8 anzulegen! Hierzu muss der Editor auf UTF-8 eingestellt werden!

Wenn das Skript selbst mit UTF-8 kodiert ist, können beliebige **Unicode-Zeichen** in Abfragen und Ausgaben direkt angegeben werden!

### XML, Skripteinbettung

Das Skript kann auch in ein XML-Dokument eingebettet werden, um die Kompatibilität mit Editoren zu erreichen:

```
<?xml version="1.0" encoding="ASCII" standalone="yes" ?>
<script>

ELEMENT dokument
BEG="HALLO"
ENDE

ENTITY auml="[Ein Umlaut 'ä']"
</script>
```

### Syntax (Hinweis)

Die Kommentaranweisungen (*//* und */\* \*/*) sind in allen Whitespace-Bereichen zulässig. Whitespace-Bereiche sind Trennbereiche zwischen Anweisungen, die in der Regel aus Leerzeichen, Tabulatoren und Zeilenwechselln bestehen. Auch innerhalb von Anweisungen sind Whitespace-Bereiche entsprechend der Anweisungs-Syntax zulässig.

### Siehe auch

*Skript-Einstellungen, Element-Handler, Entitäten-Handler*



## ***Skript-Einstellungen***

---

Parameter zur Konfiguration des *Skriptes*.

Sollen XML- oder SGML-Daten konvertiert werden?

- Siehe SYNTAX.

Sollen alle Fehler zum Abbruch der Operation führen?

- Siehe WARNINGS.

Sollen im Dokument alle Elemente, alle Entitäten und alle Texte vollständig konvertiert werden?

- Siehe CONVERTELEMENTS,
- CONVERTENTITIES und
- CONVERTDATA.

Wie werden die passenden Daten zum *Skript* identifiziert?

- Durch das Hauptelement, siehe BASE.
- Durch einen öffentlichen Bezeichner, siehe PUBLIC.

### **Syntax**

{SYNTAX} |  
{WARNINGS } |  
{CONVERTELEMENTS} |  
{CONVERTENTITIES} |  
{CONVERTDATA} |  
{READERENCODING} |  
{WRITERENCODING} |  
{BASE} |  
{SYSTEM} |  
{PUBLIC})\*

### **Siehe auch**

*Skript*

## **Unterelement**

---

Ein Unterelement bezeichnet ein Element, welches sich im Inhalt eines anderen Elementes befindet.

Der Inhalt eines Elementes, also die Menge der Unterelemente wird als *Unterstruktur* bezeichnet.

### **Beispiel**

```
<kapitel>
  <titel>Vorwort</titel>
  <p>Das ist sehr <b>wichtig</b></p>
</kapitel>
```

Unterelemente des Elementes `kapitel` sind:

- `titel`
- `p`
- `b`

Die Elemente `titel`, `p` und `b` werden als Unterelemente des Elementes `kapitel` bezeichnet. Das Element `b` ist ein Unterelement des Elementes `p`.

### **Siehe auch**

*Unterstruktur*



## **Unterstruktur**

---

Der Inhalt eines Elementes, also die Menge aller Unterelemente einschließlich der Zeichendaten und *Entitäten* wird als *Unterstruktur* bezeichnet.

### **Beispiel**

```
<kapitel>
  <titel>Vorwort</titel>
  <p>Das ist sehr <b>wichtig</b></p>
</kapitel>
```

Die Unterstruktur des Elementes `p` sind alle Daten:

Zeichendaten                    "Das ist sehr "

*Unterelement*                    `b`

mit Zeichendaten                "wichtig"

### **Siehe auch**

*Unterelement*



## **Verzweigung**

---

Eine Verzweigung steuert die Programmausführung in Abhängigkeit von Bedingungen. Wird eine Bedingung erfüllt, verzweigt sich der Programmausführung auf einen einzelnen Zweig. Als Programmausführung gelten hier die *Element-Konvertierungsregel*. Eine Verschachtelung von Verzweigungen ist in beliebiger Tiefe möglich.

### **Syntax**

IF *Bedingungsausdruck*

*Element-Konvertierungsregel*

{ELSEIF *Bedingungsausdruck*

*Element-Konvertierungsregel*}\*

{ELSE

*Element-Konvertierungsregel*}?

ENDIF

*Bedingungsausdruck* ein gültiger Bedingungsausdruck, der mit Hilfe der Abfrage- und Kontextfunktionen aufgebaut wird. Entsprechend dem Wahrheitswert der Bedingung erfolgt eine Verzweigung zu den untergeordneten Ausgabefunktionen.

*Ausgabe* eine oder mehrere Ausgabefunktionen oder auch weitere Verzweigungen.

*IF-Zweig* Primäre Bedingung. Ist die Bedingung 1 wahr, wird nur der Zweig Ausgabe 1 ausgeführt und danach zum Ende der Verzweigung gegangen (ENDIF).

*ELSEIF-Zweige* Sekundäre Bedingung (ELSEIF) oder Bedingungen. Die sekundären Bedingungen werden nacheinander auf ihren Wahrheitsgehalt geprüft. Ist eine der Bedingungen 2 ... n wahr, wird nur der betreffende Zweig (einer von Ausgabe 2 ... n) ausgeführt und danach zum Ende der Verzweigung gegangen (ENDIF).

*ELSE-Zweig* Alternativer (ELSE-)Zweig. Wenn keine der vorangegangenen Bedingungen wahr sind, wird der alternative Zweig Ausgabe m ausgeführt. Danach wird zum Ende der Verzweigung gegangen (ENDIF).

### **Siehe auch**

IF, ELSEIF, ELSE, ENDIF, BEG, END, ERR, *Element-Ausgabeanweisung*, *Element-Konvertierungsregel*





## **Zeichenkette**

---

Eine Zeichenkette (literal) stellt eine Zeichenfolge dar.

### Syntax 1

**"Zeichenfolge"**

### Syntax 2

**'Zeichenfolge2'**

*Zeichenfolge*

Eine beliebige Zeichenfolge. Das Begrenzungszeichen (") ist unzulässig.

*Zeichenfolge2*

Alternative Zeichenfolge. Das alternative Begrenzungszeichen (') ist unzulässig.

### Hinweis

Ein Unterschied in der Behandlung von Zeichenketten in den Daten und im Skript muss aber beachtet werden:

XML und SGML erlauben die Einbettung von Entitäten in Zeichenketten.

Im Skript dürfen aber keine Entitäten in Zeichenketten eingebettet werden.

### Hinweis zu XML

Die maximale Länge einer Zeichenkette ist in XML unbeschränkt. Abweichend vom Standard ist hier die maximale Länge einer Zeichenkette auf **2048 Zeichen** beschränkt worden. Bitte beachten Sie diese Einschränkung!

### Hinweis zu SGML

Die zu verwendenden Zeichen für eine Zeichenkette sind in der SGML-Deklaration festgelegt. Es wird die Reference Concrete Syntax von SGML vorausgesetzt, deren Konventionen fest integriert sind. Die maximale Länge einer Zeichenkette ist hier auf **2048 Zeichen** beschränkt.



## Zeichenkette, eingeschränkte

---

Eine eingeschränkte Zeichenkette (minimum literal) ist eine Zeichenkette, die nur aus einem eingeschränkten Zeichenbereich bestehen darf.

### Syntax 1

"Zeichenfolge"

### Syntax 2

'Zeichenfolge2'

**Zeichenfolge** Eine Zeichenfolge. Das Begrenzungszeichen (") ist unzulässig.

**Zeichenfolge2** Alternative Zeichenfolge. Das alternative Begrenzungszeichen (') ist unzulässig.

Die Zeichenfolge darf aus folgenden Zeichen bestehen:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
0123456789
RE (10)
RS (13)
SPACE (32)
()+,./:=?`
```

Folgende Zeichen sind unzulässig:

```
!#$%&*;@[\\]^_`{|}~
```

Bitte beachten Sie, dass alle Leerräume in der Zeichenkette auf einen (1) Leerraum verkürzt werden (" " wird zu " "). Dazu zählen auch Zeilenwechsel.

Alle Leerräume am Beginn und am Ende werden entfernt!

### Hinweis zu XML

Die maximale Länge einer Zeichenkette ist in XML unbeschränkt. Abweichend vom Standard ist hier die maximale Länge einer Zeichenkette auf **2048 Zeichen** beschränkt worden. Bitte beachten Sie diese Einschränkung!

### Hinweis zu SGML

Die maximale Länge einer Zeichenkette ist hier auf **2048 Zeichen** beschränkt.



# Sprachreferenz

Alphabetische Übersicht aller Anweisungen

..

Eine *Zeichenkette* (literal) stellt eine Zeichenfolge dar, die durch das Begrenzungszeichen (") gekennzeichnet wird.

## Syntax

### "Zeichenfolge"

*Zeichenfolge* Eine beliebige Zeichenfolge. Das Begrenzungszeichen (") ist unzulässig.

## Hinweis

Wenn innerhalb der *Zeichenkette* das Begrenzungszeichen (") verwendet werden soll, muss die alternative *Zeichenkette* (') verwendet werden.

## Syntaxbeispiele

```
BEG="<p>Beispiel</p>"
```

```
BEG="<p class='hinweis'>Hinweis</p>"
```

```
IF [ausrichtung="rechts"]
```

## Beispiel 1

```
BEG="<div>Abschnitt</div>"
```

In diesem Beispiel wird mit der Zeichenkette die Ausgabe des Elementes `div` mit dem Inhalt `Abschnitt` definiert. Ausgabe:

```
<div>Abschnitt</div>
```

## Beispiel 2

```
BEG="<div>'wichtig'</div>"
```

Erweitert zum Beispiel 1 wird hier das einfache Anführungszeichen (') im Text verwendet. Das doppelte Anführungszeichen (") ist hier aber nicht erlaubt. Ausgabe:

```
<div>'wichtig'</div>
```

## Beispiel 3

```
BEG="<div>"+CHR(34)+"wichtig"+CHR(34)+"</div>"
```

Wenn alternativ zum Beispiel 2 doppelte Anführungszeichen statt einfacher Anführungszeichen verwendet werden sollen, müssen diese Zeichen



außerhalb der Zeichenkette, z.B. mit der CHR-Funktion definiert werden.

Ausgabe:

```
<div>"wichtig"</div>
```

Eine weitere Möglichkeit das Anführungszeichen selbst auszugeben besteht in der Verwendung einer alternativen Zeichenkette ( ' ' ).

**Siehe auch**

'', *Zeichenkette*

..

Eine alternative *Zeichenkette* (literal) stellt eine Zeichenfolge dar, die durch das alternative Begrenzungszeichen ( ' ) gekennzeichnet wird.

### Syntax

'*Zeichenfolge*'

*Zeichenfolge* Eine beliebige Zeichenfolge. Das alternative Begrenzungszeichen ( ' ) ist unzulässig.

### Hinweis

Wenn innerhalb der alternativen *Zeichenkette* das Begrenzungszeichen( ' ) verwendet werden soll, muss die *Zeichenkette* ( " ") verwendet werden.

### Syntaxbeispiele

```
BEG='<p>Beispiel</p>'
```

```
BEG='<p class="hinweis">Hinweis</p>'
```

```
IF [ausrichtung='rechts']
```

### Beispiel 1

```
BEG='<div>Abschnitt</div>'
```

In diesem Beispiel wird mit der Zeichenkette die Ausgabe des Elementes `div` mit dem Inhalt `Abschnitt` definiert. Ausgabe:

```
<div>Abschnitt</div>
```

### Beispiel 2

```
BEG='<div>"wichtig"</div>'
```

Erweitert zum Beispiel 1 wird hier das doppelte Anführungszeichen ( " ) im Text verwendet. Das doppelte Anführungszeichen ( " ) ist hier aber nicht erlaubt. Ausgabe:

```
<div>"wichtig"</div>
```

### Beispiel 3

```
BEG='<div>'+CHR(39)+'wichtig'+CHR(39)+'</div>'
```

Wenn alternativ zum Beispiel 2 einfache Anführungszeichen statt doppelter Anführungszeichen verwendet werden sollen, müssen diese Zeichen außerhalb der Zeichenkette, z.B. mit der `CHR`-Funktion definiert werden. Ausgabe:

```
<div>'wichtig'</div>
```



Eine weitere Möglichkeit das einfache Anführungszeichen selbst auszugeben besteht in der Verwendung der Zeichenkette (" ") statt der alternativen Zeichenkette.

**Siehe auch**

" ", *Zeichenkette*

---

**[ ]**

Attributkennung. Zur Kennzeichnung von Attributen werden im Skript generell die "eckigen Klammern" [...] verwendet.

Diese Kennung wird auch in den Bedingungen (*Attributbedingung*, *Bedingungsausdruck*) für die Abfrage von Attributwerten verwendet (siehe Syntax 3).

**Syntaxbeispiele**

```
IF [ausrichtung="rechts"]
IF IN liste[typ="punkt"]
BEG=VALUE ([ausrichtung])
BEG=VALUE (liste[typ])
```

**Syntax 1****[Attributname]**

Gibt den Wert des angegebenen Attributes aus.

Diese Syntax wird innerhalb der Ausgabefunktionen FILE, IMPLIED und VALUE verwendet.

**Syntax 2****[Attributname=Vorgabewert]**

<i>Attributname</i>	<i>Bezeichner</i> für den Namen eines <i>Attributes</i> .
<i>Vorgabewert</i>	<i>Zeichenkette</i> , die den Standard- <i>Attributwert</i> definiert. Der Standardwert wird immer dann verwendet, wenn innerhalb der Instanz kein Wert für das Attribut angegeben wurde.

Definiert den Vorgabewert für das angegebene Attribut.

Diese Syntax wird innerhalb der ATTLIST-Definition verwendet.

**Syntax 3****[Attributname Operator Wert]**

<i>Attributname</i>	<i>Bezeichner</i> für den Namen eines <i>Attributes</i> .
<i>Operator</i>	Vergleichsoperator:
<	das Attribut ist kleiner als der Wert*
>	das Attribut ist größer als der Wert*
<=	das Attribut ist kleiner gleich dem Wert*
>=	das Attribut ist größer gleich dem Wert*

<b>&lt;&gt;</b>	das Attribut ist ungleich dem Wert (unabhängig der Groß- und Kleinschreibung)*
<b>=</b>	das Attribut ist gleich dem Wert (unabhängig der Groß- und Kleinschreibung)*
<b>==</b>	das Attribut ist identisch dem Wert (mit Beachtung der Groß- und Kleinschreibung)
<b>!=</b>	das Attribut ist nicht identisch mit dem Wert (mit Beachtung der Groß und Kleinschreibung)
<b>=^</b>	das Attribut beginnt mit dem Wert (unabhängig der Groß- und Kleinschreibung)
<b>==^</b>	das Attribut beginnt mit dem Wert (mit Beachtung der Groß- und Kleinschreibung)
<b>=\$</b>	das Attribut endet mit dem Wert (unabhängig der Groß- und Kleinschreibung)
<b>==\$</b>	das Attribut endet mit dem Wert (mit Beachtung der Groß- und Kleinschreibung)
<b>=*</b>	das Attribut enthält den Wert (unabhängig der Groß- und Kleinschreibung)
<b>==*</b>	das Attribut enthält den Wert (mit Beachtung der Groß- und Kleinschreibung)
<b>=~</b>	das Attribut wird gegen den angegebenen reguläre Ausdruck geprüft (unabhängig der Groß- und Kleinschreibung)
<b>==~</b>	das Attribut wird gegen den angegebenen regulären Ausdruck geprüft (unter Beachtung der Groß- und Kleinschreibung)

*Wert*

*Zeichenkette*, für den *Attributwert*.

Führt einen Vergleich zwischen dem angegebenen Attribut und der angegebenen Zeichenkette aus und gibt den Wahrheitswert zurück.

\* Wenn beide Vergleichswerte Dezimalzahlen enthalten, wird mit den Operatoren **<**, **>**, **<=**, **>=**, **<>** und **=** ein numerischer Vergleich durchgeführt. Die numerischen Vergleichswerte werden hierbei automatisch vor dem Vergleich

auf 3 Kommastellen gerundet. Folgt den numerischen Werten eine Einheit, dann wird der numerische Vergleich ausgeführt, wenn die Einheit beider Vergleichswerte unabhängig der Groß-/Kleinschreibung gleich ist. Folgende Einheiten:

- pt
- em (wird ersatzweise mit 10pt berechnet)
- mm
- cm
- in

werden erkannt. Wenn beide Vergleichswerte jeweils eine der angegebenen Einheiten enthalten, werden die Vergleichswerte intern in pt umgerechnet und numerisch verglichen.

Diese Syntax wird innerhalb von *Attributbedingung* verwendet.

#### Syntax 4

**[Attributname1 Operator Attributname2]**

*Attributname1*      *Bezeichner* für den Namen eines *Attributes*. Es wird das Attribut des angegebenen Elements der Abfrage ausgewertet.

*Attributname2*      *Bezeichner* für den Namen eines *Attributes*. Es wird das Attribut des aufrufenden Elementes (Element-Handler) ausgewertet.

*Operator*            Vergleichsoperator:

- <            kleiner\*
- >            größer\*
- <=          kleiner gleich\*
- >=          größer gleich\*
- <>          ungleich (unabhängig der Groß- und Kleinschreibung)\*
- =            gleich (unabhängig der Groß- und Kleinschreibung)\*
- ==          identisch (mit Beachtung der Groß- und Kleinschreibung)
- !=          nicht identisch (mit Beachtung der Groß und Kleinschreibung)

Führt einen Vergleich zwischen den angegebenen Attributen aus und gibt den Wahrheitswert zurück.

\* Wenn beide Vergleichswerte Dezimalzahlen enthalten, wird mit den Operatoren <, >, <=, >=, <> und = ein numerischer Vergleich durchgeführt. Die numerischen Vergleichswerte werden hierbei automatisch vor dem Vergleich auf 3 Kommastellen gerundet. Folgt den numerischen Werten eine Einheit,

dann wird der numerische Vergleich ausgeführt, wenn die Einheit beider Vergleichswerte unabhängig der Groß-/Kleinschreibung gleich ist. Folgende Einheiten:

- pt
- em (wird ersatzweise mit 10pt berechnet)
- mm
- cm
- in

werden erkannt. Wenn beide Vergleichswerte jeweils eine der angegebenen Einheiten enthalten, werden die Vergleichswerte intern in pt umgerechnet und numerisch verglichen.

Diese Syntax wird innerhalb von *Attributbedingung* verwendet.

## Typ

### Bedingung

#### Beispiel 1

```
BEG=VALUE ([nr])
```

Gibt den *Attributwert* des Attributs `nr` aus.

#### Beispiel 2

```
IF IN absatz[indent="2cm"]
```

Dies ist eine *Bedingung*, die prüft, ob sich das aktuelle *Element* im direkten Kontext des *Elementes* `absatz` befindet und ob das *Element* `absatz` ein *Attribut* `indent` mit dem *Attributwert* "2cm" besitzt.

#### Beispiel 3

```
IF [typ=""]
```

Diese Bedingung prüft, ob das *Attribut* `typ` leer ist. Hinweis: Vergleichsoperationen werden als Textvergleiche durchgeführt.

#### Beispiel 4

```
ELEMENT verweis  
ATTLIST [refid=""]  
BEG=COPY(kapitel[id=refid], DOC, FULL)  
ENDE
```

Das Element `verweis` ruft das Element `kapitel` auf, dessen Attribut `id` mir dem Attribut `refid` des Verweises übereinstimmt (klassischer Querverweis).

## Beispiel 5

### Daten

```
<test typ="Beginn Mitte Ende">
```

### Skript

```
IF [typ="beginn"]
```

Diese Bedingung prüft, ob das angegebne Attribut mit "beginn" unabhängig der Groß-/Kleinschreibung beginnt. Das Ergebnis ist Wahr.

### Skript

```
IF [typ=="beginn"]
```

Diese Bedingung prüft, ob das angegebne Attribut mit "beginn" unter Beachtung der Groß-/Kleinschreibung beginnt. Das Ergebnis ist Falsch.

### Skript

```
IF [typ="mitte"]
```

Diese Bedingung prüft, ob das angegebne Attribut den Text "mitte" unabhängig der Groß-/Kleinschreibung enthält. Das Ergebnis ist Wahr.

### Skript

```
IF [typ=="mitte"]
```

Diese Bedingung prüft, ob das angegebne Attribut den Text "mitte" unter Beachtung der Groß-/Kleinschreibung enthält. Das Ergebnis ist Falsch.

### Skript

```
IF [typ="$ende"]
```

Diese Bedingung prüft, ob das angegebne Attribut mit "ende" unabhängig der Groß-/Kleinschreibung endet. Das Ergebnis ist Wahr.

### Skript

```
IF [typ=="ende"]
```

Diese Bedingung prüft, ob das angegebne Attribut mit "ende" unter Beachtung der Groß-/Kleinschreibung beginnt. Das Ergebnis ist Falsch.

## Hinweis

Um einen vererbten Attributwert auszuwerten, verwenden Sie die Abfrage IMPLIED.

## Siehe auch

IMPLIED, *Attributname*, *Attributwert*

---

**//**

Beginn einer Kommentarzeile. Kennzeichnet den nachfolgenden Text bis zum Zeilenende als Kommentar.

### Syntax

**//** *Kommentar*

*Kommentar*

Kommentarzeile, bestehend aus beliebigem Text, der mit einem Zeilenwechsel (Enter, CR LF) abgeschlossen wird.

### Syntaxbeispiele

```
IF [ausrichtung="rechts"] // Ausrichtung rechts?  
IF IN liste[typ="punkt"] // in Punktliste  
BEG=VALUE([ausrichtung]) // Ausrichtung
```

### Beispiel 1

```
IF IN kapitel  
  // BEG="<TEST>" auskommentierter Befehl  
  BEG="<H1>"  
ENDIF
```

### Beispiel 2

```
IF IN kapitel  
  BEG="<H1>" // Das ist eine Ueberschrift  
ENDIF
```

### Siehe auch

*/\* \*/*, *Kommentar*

---

***/\* \*/***

Kommentarbereich.

**Syntax*****/\* Kommentar \*/******/\**** Beginn des Kommentarbereiches.***Kommentar*** beliebiger Text.***\*/*** Ende des Kommentarbereiches.**Hinweis**Die Kommentarende-Kennung (*\*/*) darf im Kommentartext nicht enthalten sein.**Syntaxbeispiele**

```
// komplett auskommentierter Skript-Block:  
/* IF [ausrichtung="rechts"]  
    BEG='<p class="rechts">Rechts</p>  
ENDIF  
*/  
// auskommentierter Skript-Bereich:  
IF /* IN beitrage */ IN kapitel
```

**Beispiel 1**

```
/* Beginn der Auskommentierung  
IF IN kapitel  
    BEG="<H1>"  
ENDIF  
Ende der Auskommentierung */
```

**Beispiel 2**

```
IF IN /* Kommentar */ kapitel  
    BEG="<H1>"  
ENDIF
```

**Siehe auch***//, Kommentar*

## AFTER

---

Prüft, ob sich das aktuelle *Element* direkt hinter dem angegebenen *Element* befindet.

### Syntax 1

**AFTER** {*Elementname*}{*Attributbedingung*}?

### Syntax 2

**AFTER**({*Elementname*}{*Attributbedingung*}?)

*Elementname*                    *Bezeichner* für ein *Element*.

*Attributbedingung*            [*Attributname Operator Zeichenkette*]  
*Bedingung*, die einen Wert eines *Attributs* abfragt.  
Syntax hierfür siehe *Attributbedingung*.

Diese Abfrage liefert dann *Richtig*, wenn sich das aktuelle *Element* direkt hinter dem angegebenen Element befindet.

Durch Angabe einer *Attributbedingung* kann diese für das angegebene Element geprüft werden. Die Abfrage liefert dann *Richtig*, wenn sich das aktuelle Element direkt hinter dem angegebenen Element befindet und die *Attributbedingung* für das angegebene Element erfüllt wird.

### Hinweis

Wird als Parameter einer Abfrage ein Element **ohne Klammern** angegeben und liefert diese Abfrage *Richtig*, dann wird dieses Element für die folgende Abfrage als das aktuelle Element betrachtet. Das Element des Parameters wird also zum aktuellen Element erklärt!

Wird als Parameter einer Abfrage ein Element **in Klammern** angegeben, bleibt das aktuelle Element unverändert.

### Syntaxbeispiele

```
IF AFTER titel
IF AFTER beitrags
IF IN eintrag AFTER eintrag
IF INX kapitel AFTER vorwort
IF NOTAFTER titel
IF NOTAFTER beitrags
IF IN eintrag NOTAFTER eintrag
IF INX kapitel NOTAFTER vorwort
IF NOTAFTER(titel)
```

## Typ

### Bedingung

#### Beispiel 1

##### Daten:

```
<dokument>
<liste>
<eintrag>Elefanten</eintrag>
<eintrag>Tiger</eintrag>
<eintrag>Affen</eintrag>
</liste>
</dokument>
```

##### Skript:

```
ELEMENT dokument
BEG="<html><body>"
END="</body></html>"
ENDE

ELEMENT liste
BEG="<ul>"
END="</ul>"
ENDE

ELEMENT eintrag
BEG="<li>"
IF AFTER eintrag
  END=" (AFTER eintrag) "
ENDIF
IF NOTAFTER eintrag
  END=" (NOTAFTER eintrag) "
ENDIF
END="</li>"
ENDE
```

##### Ergebnis:

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

```
<ul>
<li>Elefanten (NOTAFTER eintrag)</li>
<li>Tiger (AFTER eintrag)</li>
<li>Affen (AFTER eintrag)</li>
</liste>
</ul>
</body></html>
```

In diesem Beispiel wird mit `AFTER` geprüft, ob sich der jeweilige Listeneintrag `eintrag` direkt nach einem Listeneintrag `eintrag` befindet. In diesem Fall wird der Text " (AFTER eintrag)" angehängt.

## Beispiel 2

Daten:

```
<p><fn>Fu&szlig;note</fn><fn>Fu&szlig;note</fn></p>
```

Skript:

```
ELEMENT fn
IF AFTER fn
  BEG=" "
ENDIF
ENDE
```

Ergebnis:

```
Fußnote Fußnote
```

Dieses Beispiel prüft, ob die zu verarbeitende Fußnote (Element `fn`) direkt einer Fußnote folgt. In diesem Fall wird ein Leerzeichen zwischen die Fußnoten eingefügt.

## Siehe auch

BEFORE, BEG, EMPTY, END, FIND, FIRST, IN, INX, LAST, SUB, TOP-LEVEL, *Bedingungsausdruck*, *Bezeichner*, *Element*

## AFTERX

---

Prüft, ob das aktuelle *Element* ein beliebiges Nachfolgeelement vom angegebenen *Element* ist.

### Syntax 1

**AFTERX** {*Elementname*}{*Attributbedingung*}?

### Syntax 2

**AFTERX**(({*Elementname*}{*Attributbedingung*}?)

<i>Elementname</i>	<i>Bezeichner</i> für ein <i>Element</i> .
<i>Attributbedingung</i>	[ <i>Attributname Operator Zeichenkette</i> ] <i>Bedingung</i> , die einen Wert eines <i>Attributs</i> abfragt. Syntax hierfür siehe <i>Attributbedingung</i> .

Diese Abfrage liefert dann *Richtig*, wenn sich das aktuelle *Element* an beliebiger Position hinter dem angegebenen Element befindet (auf der gleichen Ebene wie das aktuelle Element).

Durch Angabe einer *Attributbedingung* kann diese für das angegebene Element geprüft werden. Die Abfrage liefert dann *Richtig*, wenn sich das aktuelle Element an beliebiger Position hinter dem angegebenen Element befindet und die *Attributbedingung* für das angegebene Element erfüllt wird.

### Hinweis

Wird als Parameter einer Abfrage ein Element **ohne Klammern** angegeben und liefert diese Abfrage *Richtig*, dann wird dieses Element für die folgende Abfrage als das aktuelle Element betrachtet. Das Element des Parameters wird also zum aktuellen Element erklärt!

Wird als Parameter einer Abfrage ein Element **in Klammern** angegeben, bleibt das aktuelle Element unverändert.

### Syntaxbeispiele

```
IF AFTERX titel
IF AFTERX beitrags
IF IN eintrag AFTERX eintrag
IF INX kapitel AFTERX vorwort
IF NOTAFTERX titel
IF NOTAFTERX beitrags
IF IN eintrag NOTAFTERX eintrag
IF INX kapitel NOTAFTERX vorwort
IF NOTAFTERX (titel)
```

## Typ

### *Bedingung*

### Beispiel

#### Daten:

```
<dokument>
<kapitel>
<titel>Wichtige Information</titel>
<text>Das ist der Text zum Kapitel mit Titel</text>
</kapitel>
<kapitel>
<text>Das ist der Text zum Kapitel ohne Titel</text>
</kapitel>
</dokument>
```

#### Skript:

```
ELEMENT dokument
BEG="<html><body>"
END="</body></html>"
ENDE
```

```
ELEMENT kapitel
BEG="<br>"
ENDE
```

```
ELEMENT titel
BEG="<h1>"
END="</h1>"
ENDE
```

```
ELEMENT text
BEG="<p>"
IF AFTERX titel
  BEG="<i>"
  END="</i>"
ENDIF
```

```
END="</p>"
```

```
ENDE
```

**Ergebnis:**

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

```
<br>
```

```
<h1>Wichtige Information</h1>
```

```
<text><i>Das ist der Text zum Kapitel mit Titel</i></text>
```

```
<br>
```

```
<text>Das ist der Text zum Kapitel ohne Titel</text>
```

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

In diesem Beispiel wird mit `AFTERX` geprüft, ob sich der jeweilige Text in einem Kapitel mit `titel` befindet. In diesem Fall wird der Text kursiv dargestellt angehängt.

**Siehe auch**

`BEFORE`, `BEG`, `EMPTY`, `END`, `FIND`, `FIRST`, `IN`, `INX`, `LAST`, `SUB`, `TOP-LEVEL`, *Bedingungsausdruck*, *Bezeichner*, *Element*

## ALTERNATE

---

Definiert eine alternative Entitäten-Ersetzung.

Diese Ersetzungsregeln können zusätzlich zur Entitäten-Definition im Skript eingefügt werden, um eine alternative Umsetzung von Entitäten zu realisieren. Diese Deklaration sollte für eine ASCII-konforme Umsetzung bzw. Umschreibung der Entitäten verwendet werden.

### Syntax

**ALTERNATE** = *Ausgabeeanweisung* {+ *Ausgabeeanweisung* }\*

*Ausgabeeanweisung*     Eine *Entitäten-Ausgabeeanweisung*. Anweisung, deren Ergebnis an dieser Position ausgegeben wird.

**+**     Verkettungsoperator. Mit diesem Operator werden Texte oder *Entitäten-Ausgabeeanweisungen* zusammengeführt. Die Ausgabe erfolgt in Reihenfolge der Verkettung von links nach rechts.

Die alternative Konvertierung wird mit dem Datentyp DAT=ALT oder mit dem Befehl COPY und dem Ersetzungstyp **ALT** aufgerufen. Ist für eine Entität keine alternative Deklaration vorhanden, wird die Standard-Deklaration (ENTITY) verwendet.

### Hinweis

Diese Anweisung ist Bestandteil der ENTITY-Anweisung.

### Syntaxbeispiele

```
ENTITY auml("&auml;" ALTERNATE="ae"  
ENTITY auml("&uuml;" ALTERNATE="ue" ASCII="ü"
```

### Typ

#### Anweisung

#### Beispiel

##### Daten:

```
<dokument>  
<p>Steuern<index>Geb&uuml;hren</index></p>  
</dokument>
```

##### Skript:

```
ELEMENT dokument  
BEG='<meta name="keywords" content="'
```

```
BEG=COPY(index, ONE, ALT)
BEG=' '>'
ENDE
ENTITY auml="&uuml" ALTERNATE="ue"
```

**Ausgabe:**

```
<meta name="keywords" content="Gebuehren">
```

In diesem Beispiel wird die Entität `uuml` standardmäßig unverändert als HTML-Entität `&uuml` konvertiert. Bei einem alternativen Ersetzungsmodell wird diese Entität dann alternativ mit `ue` umschrieben.

**Hinweis**

Rekursionsschutz! Bei Verwendung von `VALUE` innerhalb einer *Entitäten-Ausgabeangabe* werden nur Textbestandteile (CDATA) des betreffenden *Attributwerts* zurückgegeben und wiederum enthaltene Entitäten ausgelassen!

**Siehe auch**

ENTITY, ASCII, *Entitäten-Handler*, *Entitäten-Konvertierungsregel*

**Benötigt von**

COPY, IMPLIED, VALUE

## ASCII

---

Definiert die ASCII-Entitäten-Ersetzung.

Diese Ersetzungsregeln können zusätzlich zur Entitäten-Definition im Skript eingefügt werden, um eine alternative Umsetzung von Entitäten zu realisieren. Diese Deklaration sollte für eine ASCII-konforme Umsetzung bzw. Umschreibung der Entitäten verwendet werden.

### Syntax

**ASCII** = *Ausgabeanweisung* {+ *Ausgabeanweisung* }\*

*Ausgabeanweisung*     Eine *Entitäten-Ausgabeanweisung*. Anweisung, deren Ergebnis an dieser Position ausgegeben wird.

**+**     Verkettungsoperator. Mit diesem Operator werden Texte oder *Entitäten-Ausgabeanweisungen* zusammengeführt. Die Ausgabe erfolgt in Reihenfolge der Verkettung von links nach rechts.

Die alternative ASCII-Konvertierung wird mit dem Datentyp DAT=ASC oder mit dem Befehl COPY und dem Ersetzungstyp **ASC** aufgerufen. Ist für eine Entität keine ASCII-Deklaration vorhanden, wird die Standard-Deklaration (ENTITY) verwendet.

### Syntaxbeispiele

```
ENTITY auml="&auml;" ASCII="ä"
ENTITY auml="&uuml;" ALTERNATE="ue" ASCII="ü"
```

### Typ

*Anweisung*

### Beispiel

Daten:

```
<dokument>
<p>Steuern<index>Geb&uuml;hren</index></p>
</dokument>
```

Skript:

```
ELEMENT dokument
BEG='<meta name="keywords" content="'
BEG=COPY(index, ONE, ALT)
BEG=' ">'
ENDE
```

```
ENTITY auml="&uuml" ALTERNATE="ue" ASCII="ü"
```

**Ausgabe:**

```
<meta name="keywords" content="Gebühren">
```

In diesem Beispiel wird die Entität `uuml` standardmäßig unverändert als HTML-Entität `&uuml` konvertiert. Bei einem alternativen ASCII-Ersetzungsmodell wird diese Entität dann mit `ü` umschrieben.

**Hinweis**

Rekursionsschutz! Bei Verwendung von VALUE innerhalb einer *Entitäten-Ausgabeweisung* werden nur Textbestandteile (CDATA) des betreffenden *Attributwerts* zurückgegeben und wiederum enthaltene Entitäten ausgelassen!

**Siehe auch**

ENTITY, ALTERNATE, *Entitäten-Handler*, *Entitäten-Konvertierungsregel*

**Benötigt von**

COPY, IMPLIED, VALUE

## ATTLIST

---

Definiert die Vorgabewerte für die Attribute eines Elementes.

### Syntax

**ATTLIST** [*Name=Wert*]{ [*Name=Wert*]}\*

*Name*                                *Bezeichner* für den *Attributname* eines *Attributes*.

*Wert*                                    *Zeichenkette*, die den Vorgabewert für das angegebene Attribut.

Als Vorgabewerte für das Skript sollten die in der DTD angegebenen Vorgabewerte übernommen werden.

### Hinweis

Da die DTD nicht ausgewertet wird, sollten die Vorgabewerte unbedingt in das Skript übernommen werden.

### Syntaxbeispiele

```
ATTLIST [typ="punkt] [id=""]  
ATTLIST [ausrichtung="links"]
```

### Typ

*Anweisung*

### Beispiel

```
ELEMENT kapitel  
ATTLIST [id=""] [toc="0"]  
ENDE
```

Für das *Element* `kapitel` werden das Attribut `id` ohne Vorgabewert und das Attribut `toc` mit Vorgabewert `0` definiert.

### Siehe auch

ELEMENT, [, *Attributname*, *Attributwert*, *Element-Handler*, *Skript*, *Zeichenkette*

## BASE

---

Definiert das Basiselement des XML-/SGML-Dokumentes.

Mit diesem Befehl veranlassen Sie die Prüfung des Basiselementes der Instanzen. Durch Angabe eines oder mehrerer Basiselemente im Skript wird die Prüfung aktiviert. Instanzen mit einem nicht definierten Basiselement werden nicht konvertiert.

### Syntax

**BASE** = *Elementname*

*Elementname*                      *Zeichenkette*, die den *Bezeichner* für ein *Element* enthält. Das Basiselement ist das oberste Element innerhalb eines Dokumentes.

### Syntaxbeispiele

```
BASE="dokument"  
BASE="kapitel"  
BASE="beispiel"
```

### Typ

*Anweisung*

### Beispiel 1

Daten:

```
<dokument>  
</dokument>
```

Skript:

```
BASE = "dokument"
```

Instanzen werden nur dann konvertiert, wenn das Basiselement der Instanz mit einem angegebenen Basiselement übereinstimmt.

### Beispiel 1

Dokument A:

```
<!DOCTYPE kapitel SYSTEM "meine.dtd">  
<p>  
</p>
```

Dokument B:

```
<!DOCTYPE beispiel SYSTEM "meine.dtd">  
<p>  
</p>
```



Skript:

**BASE** = "kapitel"

Dokument A wird konvertiert, da das Basiselement des Dokumentes mit dem definierten Basiselement übereinstimmt. Das Dokument B wird nicht konvertiert. Es wird eine Fehlermeldung ausgegeben.

**Siehe auch**

*Skript, Skript-Einstellungen, Bezeichner, Element*

## BEFORE

---

Prüft, ob sich das aktuelle *Element* direkt vor dem angegebenen *Element* befindet.

### Syntax 1

**BEFORE** {*Elementname*}{*Attributbedingung*}?

### Syntax 2

**BEFORE**({*Elementname*}{*Attributbedingung*}?)

*Elementname*                    *Bezeichner* für ein *Element*.

*Attributbedingung*            *Bedingung*, die einen Wert eines *Attributs* abfragt.  
Syntax hierfür siehe *Attributbedingung*.

Diese Abfrage liefert dann *Richtig*, wenn sich das aktuelle *Element* direkt vor dem angegebenen Element befindet.

Durch Angabe einer *Attributbedingung* kann diese für das angegebene Element geprüft werden. Die Abfrage liefert dann *Richtig*, wenn sich das aktuelle Element direkt vor dem angegebenen Element befindet und die *Attributbedingung* für das angegebene Element erfüllt wird.

### Hinweis

Wird als Parameter einer Abfrage ein Element **ohne Klammern** angegeben und liefert diese Abfrage *Richtig*, dann wird dieses Element für die folgende Abfrage als das aktuelle Element betrachtet. Das Element des Parameters wird also zum aktuellen Element erklärt!

Wird als Parameter einer Abfrage ein Element **in Klammern** angegeben, bleibt das aktuelle Element unverändert.

### Syntaxbeispiele

```
IF BEFORE anhang
IF BEFORE (anhang)
IF IN p BEFORE liste
IF NOTBEFORE anhang
IF IN p NOTBEFORE liste
IF IN p NOTBEFORE (liste)
```

### Typ

*Bedingung*

## Beispiel 1

### Daten:

```
<dokument>
<liste>
<eintrag>Elefanten</eintrag>
<eintrag>Tiger</eintrag>
<eintrag>Affen</eintrag>
</liste>
</dokument>
```

### Skript:

```
ELEMENT dokument
BEG="<html><body>"
END="</body></html>"
ENDE

ELEMENT liste
BEG="<ul>"
END="</ul>"
ENDE

ELEMENT eintrag
BEG="<li>"
IF BEFORE eintrag
  END=" (BEFORE eintrag) ""
ENDIF
IF NOTBEFORE eintrag
  END=" (NOTBEFORE eintrag) ""
ENDIF
END="</li>"
ENDE
```

### Ergebnis:

```
<html><body>
<ul>
<li>Elefanten (BEFORE eintrag)</li>
<li>Tiger (BEFORE eintrag)</li>
<li>Affen (NOTBEFORE eintrag)</li>
</liste>
```

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

In diesem Beispiel wird mit `BEFORE` geprüft, ob sich der jeweilige Listeneintrag `eintrag` direkt vor einem Listeneintrag `eintrag` befindet. In diesem Fall wird der Text " (`BEFORE eintrag`)" angehängt.

### Beispiel 2

```
ELEMENT kapitel
IF BEFORE kapitel
  BEG="Ich befinde mich vor einem Kapitel"
ENDIF
ENDE
```

### Beispiel 3

```
ELEMENT kapitel
ATTLIST [typ="textbereich"]
IF BEFORE kapitel[typ="anhang"]
  BEG="Ich befinde mich vor dem Anhang"
ENDIF
ENDE
```

### Beispiel 4

```
ELEMENT fn
IF BEFORE fn
  END=" "
ENDIF
ENDE
```

Dieses Beispiel prüft, ob die zu verarbeitende Fußnote (Element `fn`) sich direkt vor einer Fußnote befindet. In diesem Fall wird ein Leerzeichen zwischen den Fußnoten eingefügt.

### Siehe auch

`AFTER`, `BEG`, `EMPTY`, `END`, `FIND`, `FIRST`, `IN`, `INX`, `LAST`, `SUB`, `TOPLEVEL`, *Bedingungsausdruck*, *Bezeichner*, *Element*

## BEFOREX

---

Prüft, ob sich das aktuelle *Element* ein beliebiges Vorgängerelement vom angegebenen *Element* ist.

### Syntax 1

**BEFOREX** {*Elementname*}{*Attributbedingung*}?

### Syntax 2

**BEFOREX**{(*Elementname*){*Attributbedingung*}?)

*Elementname*                    *Bezeichner* für ein *Element*.

*Attributbedingung*            *Bedingung*, die einen Wert eines *Attributs* abfragt.  
Syntax hierfür siehe *Attributbedingung*.

Diese Abfrage liefert dann *Richtig*, wenn sich das aktuelle *Element* an beliebige Position vor dem angegebenen Element befindet (auf gleicher Ebene wie das aktuelle Element).

Durch Angabe einer *Attributbedingung* kann diese für das angegebene Element geprüft werden. Die Abfrage liefert dann *Richtig*, wenn sich das aktuelle Element an beliebiger Position vor dem angegebenen Element befindet und die *Attributbedingung* für das angegebene Element erfüllt wird.

### Hinweis

Wird als Parameter einer Abfrage ein Element **ohne Klammern** angegeben und liefert diese Abfrage *Richtig*, dann wird dieses Element für die folgende Abfrage als das aktuelle Element betrachtet. Das Element des Parameters wird also zum aktuellen Element erklärt!

Wird als Parameter einer Abfrage ein Element **in Klammern** angegeben, bleibt das aktuelle Element unverändert.

### Syntaxbeispiele

```
IF BEFOREX anhang
IF BEFOREX (anhang)
IF IN p BEFOREX liste
IF NOTBEFOREX anhang
IF IN p NOTBEFOREX liste
IF IN p NOTBEFOREX(liste)
```

### Typ

*Bedingung*

**Beispiel****Daten:**

```
<dokument>
<kapitel>
<titel>1. Einleitung</titel>
<text>Das ist der Text zum Kapitel Einleitung</text>
</kapitel>
<kapitel>
<titel>2. Grundlagen</titel>
<text>Das ist der Text zum Kapitel Grundlagen</text>
</kapitel>
<anhang>
<titel>Anhang</titel>
<text>Das ist der Text zum Anhang</text>
</anhang>
</dokument>
```

**Skript:**

```
ELEMENT dokument
BEG="<html><body>"
END="</body></html>"
ENDE
```

```
ELEMENT kapitel
IF BEFOREX anhang
  IF FIRST
    BEG="<div>"
  ELSEIF LAST
    END="</div>"
  ENDIF
ENDIF
ENDE
```

```
ELEMENT anhang
IF AFTER kapitel
  BEG="<div>"
  END="</div>"
ENDIF
```

ENDE

ELEMENT titel

BEG="<h1>"

END="</h1>"

ENDE

ELEMENT text

BEG="<p>"

END="</p>"

ENDE

### Ergebnis:

```
<html><body>
<div>
<h1>1. Einleitung</h1>
<p>Das ist der Text zum Kapitel Einleitung</p>
<h1>2. Grundlagen</h1>
<p>Das ist der Text zum Kapitel Grundlagen</p>
</div>
<div>
<h1>Anhang</h1>
<p>Das ist der Text zum Anhang</p>
</div>
</body></html>
```

In diesem Beispiel wird mit `BEFOREX` geprüft, ob sich auf Ebene der Kapitel ein Anhang befindet und in diesem Fall werden alle Kapitel in einem `div` zusammengefasst und auch der Anhang in ein `div` strukturiert.

### Siehe auch

AFTER, BEG, EMPTY, END, FIND, FIRST, IN, INX, LAST, SUB, TOPLEVEL, *Bedingungsausdruck*, *Bezeichner*, *Element*

## BEG

Verfügbar als Anweisung (siehe BEG-Anweisung) und als Bedingung (siehe BEG-Bedingung).

### BEG-Anweisung

Definiert den Ersetzungstext für den Starttag. Bei Mehrfachdefinitionen werden diese in Reihenfolge der Ausführung im Skript umgesetzt.

Der Ersetzungstext kann durch eine beliebige Kombination von *Zeichenketten* und Makrofunktionen definiert werden, wobei diese mit dem "+"-Operator zu verbinden sind.

### Syntax

**BEG** = *Ausgabeanweisung* {+ *Ausgabeanweisung* }\*

*Ausgabeanweisung*     Eine *Element-Ausgabeanweisung*. Anweisung, deren Ergebnis an dieser Position ausgegeben wird.

**+**     Verkettungsoperator. Mit diesem Operator werden Texte oder *Element-Ausgabeanweisungen* zusammengeführt. Die Ausgabe erfolgt in Reihenfolge der Verkettung von links nach rechts.

### Syntaxbeispiele

```
BEG='<p>Beispiel</p>'  
BEG='<p class="hinweis">Hinweis</p>'  
BEG='<p>Beispiel</p>'+CHR(13)+"<p>...</p>"  
BEG='<p>Beispiel: '+VALUE([nr])+"</p>"  
// typisches Elementumsetzung:  
BEG="<p>"  
END="</p>"
```

### Typ

*Anweisung*

#### Beispiel 1

```
ELEMENT p  
BEG="<div>"  
END="</div>"  
ENDE
```

Mit der Anweisung BEG wird in diesem Beispiel statt dem Starttag <p> ein <div> ausgegeben.

**Beispiel 2**

```
ELEMENT kapitel
```

```
ATTLIST [id=""]
```

```
BEG="<a name="+VALUE([id])+>"
```

```
ENDE
```

Mit der Anweisung **BEG** wird in diesem Beispiel das Element `<a>` mit dem Attribut `name` ausgegeben. Der Attributwert von `name` wird aus dem Attribute `id` des Elementes `kapitel` übernommen. Mit den Verkettungsoperatoren wird die Ausgabe zusammengestellt.

**Siehe auch**

CHR, COPY, COUNT, CUT, DEL, END, ERR, FILE, IMPLIED, TOKEN, VALUE, *Element-Ausgabeeweisung*, *Element-Konvertierungsregel*, *Verzweigung*, *Zeichenkette*

## BEG-Bedingung

---

Prüft, ob gerade ein Starttag bearbeitet wird.

### Syntax

#### BEG

Die Bedingung BEG liefert `Wahr` bei der Bearbeitung eines Starttags. Bei der Bearbeitung eines Endtags wird `Falsch` zurückgegeben. Mit der Bedingung BEG (Abfrage) können Verzweigungen so gesteuert werden, so dass diese nur beim Auftreten eines Starttags ausgeführt werden.

### Hinweis

Da die Konvertierungsregeln für komplette Elemente definiert werden, werden alle Kontextabfragen sowohl für den Starttag als auch für den Endtag durchgeführt. Zur Laufzeit-Optimierung des Skriptes kann es erforderlich sein, diese Kontextabfragen jeweils nur für den Start- oder Endtag zuzulassen.

### Syntaxbeispiele

```
// Beispiel-Block:
IF BEG
  // Dieser Block wird nur fuer den Starttag
  // ausgefuehrt
  BEG="<p>"
  IF IN eintrag IN liste[typ=punkt]
    BEG="+ "
  ENDIF
  END="-:(" // Dieser Befehl wird nie ausgefuehrt!
ENDIF
```

### Typ

#### *Bedingung*

### Beispiel

#### Daten:

```
<kapitel id="TBB.4">
<titel>Berechnung</titel>
</kapitel>
```

#### Skript:

```
ELEMENT titel
```

IF **BEG**

```
BEG="<h1>"
```

```
IF kapitel[id<>""]
```

```
  BEG='<a name="'+VALUE(kapitel[id])+'"></a>'
```

```
ENDIF
```

ELSE

```
  END="</h1>"
```

ENDIF

ENDE

Ergebnis:

```
<h1><a name="TBB.4"></a>Berechnung</h1>
```

Die Abfrage nach dem *Attribut* `id` wird in diesem Fall nur durchgeführt, wenn ein Starttag konvertiert wird. Im Ergebnis der Ausgabe ergeben sich keine Unterschiede, allerdings in der verbesserten Laufzeit des Programms.

**Siehe auch**

AFTER, BEFORE, EMPTY, END, FIND, FIRST, IMPLIED, IN, INX, LAST, SUB, TOPLEVEL, *Bedingungsausdruck*

## CHR

---

Wandelt den angegebenen Zeichencode in das repräsentierende Zeichen um. Die Daten werden an der aktuellen Position eingefügt.

### Syntax

#### **CHR**(Zeichencode)

*Zeichencode*

eine Zahl, die ein Zeichen im ASCII Zeichensatz darstellt bzw. ein beliebiges Byte (x0 - 0xFF) oder ein Unicode-Zeichen (x0 - 0x10FFFF).

### Hinweis

Bitte verwenden Sie Zeichencodes außerhalb des ASCII-Zeichenbereiches mit größtmöglicher Vorsicht!

### Typ

*Anweisung*

### Syntaxbeispiele

```
BEG="<p>Hallo</p>" + CHR (13) + CHR (10)
```

```
BEG="<p>" + CHR (34) + "Hallo" + CHR (34) + "</p>"
```

#### Beispiel 1

```
BEG=CHR (13) + CHR (10)
```

Ausgabe eines Zeilenwechsels.

#### Beispiel 2

```
BEG=CHR (169)
```

Ausgabe des Copyrightzeichens ©.

#### Beispiel 3

```
BEG=CHR (128658)
```

Erweiterter Unicodebereich, Ausgabe des roten Feuerwehrautos .

### Siehe auch

BEG, CHR, COPY, COUNT, CUT, DEL, END, ERR, FILE, IMPLIED, TOKEN, VALUE, *Element-Ausgabeanweisung*, *Entitäten-Ausgabeanweisung*, *Zeichenkette*

## CONVERTDATA

---

Steuert die Konvertierung der Zeichendaten. Mit diesem Schalter lässt sich die Datenkonvertierung der Zeichendaten (PCDATA) einschränken. Dazu müssen im *Skript* die Inhaltsmodelle (MODEL) der Elemente definiert werden.

### Syntax

**CONVERTDATA** = {**ALL**|**ONLYDEFINED**}

ALL	Alle Zeichendaten werden ausgegeben. ( <b>Standard</b> )
ONLYDEFINED	Nur die Datenbereiche, die als PCDATA definiert sind, werden ausgegeben. Die Definition muss dazu in der Elementdefinition ELEMENT mit dem Befehl MODEL erfolgen. Alle Inhalte vom Typ DATA (entspricht #PCDATA oder #RCDATA) und MIXED (entspricht einem Inhaltsmodell, welches Elemente und #PCDATA enthalten kann) werden ausgegeben. Alle Datenbereiche anderer Modelle werden ausgelassen. Ist kein Datenmodell definiert, wird MIXED angenommen und die Inhalte ausgegeben.

### Syntaxbeispiele

```
CONVERTDATA=ALL
```

```
CONVERTDATA=ONLYDEFINED
```

### Typ

*Anweisung*

#### Beispiel 1

```
CONVERTDATA=ALL
```

Es werden alle in den Daten vorkommenden Zeichendaten ausgegeben.

#### Beispiel 2

```
CONVERTDATA=ONLYDEFINED
```

Es werden nur von den Elementen Zeichendaten ausgegeben, die gemäß MODEL-Definition PCDATA enthalten dürfen. Alle anderen Inhalte werden nicht ausgegeben!

### Siehe auch

CONVERTELEMENTS, CONVERTENTITIES, *Skript*, *Skript-Einstellungen*

## CONVERTELEMENTS

---

Steuert die Konvertierung der Elemente. Mit diesem Schalter lässt sich die Elementkonvertierung auf die im *Skript* definierten Elemente (siehe *Element-Handler*) einschränken. In diesem Fall werden nicht definierte Elemente unverändert ausgegeben.

### Syntax

**CONVERTELEMENTS** = {**ALL**|**ONLYDEFINED**|**DROPUNDEFINED**}

ALL	Alle Elemente werden übersetzt, d.h., für jedes <i>Element</i> muss eine Ersetzungsregel existieren.
ONLYDEFINED	Nur die definierten Elemente werden übersetzt, alle nicht definierten Elemente werden unverändert ausgegeben. ( <b>Standard</b> )
DROPUNDEFINED	Die definierten Elemente werden übersetzt und alle nichtdefinierten Elemente werden ausgelassen.

### Syntaxbeispiele

```
CONVERTELEMENTS=ALL
```

```
CONVERTELEMENTS=ONLYDEFINED
```

```
CONVERTELEMENTS=DROPUNDEFINED
```

### Typ

*Anweisung*

#### Beispiel 1

```
CONVERTELEMENTS=ALL
```

Es werden alle in den Daten vorkommenden Elemente entsprechend den Einstellungen in den *Element-Konvertierungsregel* konvertiert.

#### Beispiel 2

```
CONVERTELEMENTS=ONLYDEFINED
```

Es werden von den in den Daten vorkommenden Elementen nur diejenigen, die in den *Element-Konvertierungsregel* definiert wurden, konvertiert. Alle anderen Elemente bleiben unverändert.

#### Beispiel 3

```
CONVERTELEMENTS=DROPUNDEFINED
```

Es werden von den in den Daten vorkommenden Elementen nur diejenigen, die in den *Element-Konvertierungsregel* definiert wurden, konvertiert. Alle anderen Elemente werden ausgelassen.



**Siehe auch**

*Skript*, *Skript-Einstellungen*, CONVERTDATA, CONVERTENTITIES



## CONVERTENTITIES

---

Steuert die Konvertierung der Entitäten. Mit diesem Schalter lässt sich die Entitätenkonvertierung auf die im *Skript* definierten Entitäten (siehe *Entitäten-Handler*) einschränken. In diesem fall werden nicht definierte Entitäten unverändert ausgegeben.

### Syntax

**CONVERTENTITIES** = {**ALL**|**ONLYDEFINED**}

ALL	Alle Entitäten werden übersetzt, d.h., für jede Entität muss eine Ersetzungsregel existieren.
ONLYDEFINED	Nur die definierten Entitäten werden übersetzt, alle nicht definierten Entitäten werden unverändert ausgegeben. ( <b>Standard</b> )

### Syntaxbeispiele

```
CONVERTENTITIES=ALL
```

```
CONVERTENTITIES=ONLYDEFINED
```

### Typ

*Anweisung*

### Beispiel 1

```
CONVERTENTITIES=ALL
```

Es werden alle in den Daten vorkommenden Entitäten entsprechend den Einstellungen in den *Entitäten-Konvertierungsregel* konvertiert.

### Beispiel 2

```
CONVERTENTITIES=ONLYDEFINED
```

Es werden von den in den Daten vorkommenden Entitäten nur diejenigen, die in den *Entitäten-Konvertierungsregel* definiert wurden, konvertiert. Alle anderen Entitäten bleiben unverändert.

### Siehe auch

CONVERTDATA, CONVERTELEMENTS, *Skript*, *Skript-Einstellungen*

## COPY

---

Sucht das erste Auftreten des angegebenen Elementes in der *Unterstruktur* des aktuellen Elementes oder im kompletten Dokument und gibt die Daten anhand der angegebenen Ersetzungsregeln zurück.

### Syntax 1

**COPY**(*Elementname*{, *Suchbereich*}?{, *Anzahl*}?, *Ersetzungstyp* {, *Kontextname*}?{, *Trenntext*}?)

### Syntax 2

**COPY**(*Elementname*{*Attributbedingung*}?{*Kontextbedingung*}?{, *Suchbereich*}?{, *Anzahl*}?, *Ersetzungstyp* {, *Kontextname*}?{, *Trenntext*}?)

### Syntax 3

**COPY**(, *Ersetzungstyp* {, *Kontextname*}?{, *Trenntext*}?)

<i>Elementname</i>	<i>Bezeichner</i> für ein <i>Element</i> . Ohne Angabe eines Elementnamens wird das aktuelle Element verwendet. Achtung, in diesem Fall ist die Verwendung eines Kontextnamens erforderlich, um eine Rekursion zu vermeiden!
<i>Attributbedingung</i>	[ <i>Attributname Operator Zeichenkette</i> ] oder [ <i>Attributname Operator Attributname</i> ] <i>Bedingung</i> , die einen Wert eines <i>Attributs</i> abfragt. Syntax hierfür siehe <i>Attributbedingung</i>
<i>Kontextbedingung</i>	eine IN, INX oder deren Negation mit dem NOT-Präfix.
<i>Suchbereich</i>	Optionale Angabe des Suchbereiches. Dieser gibt an, wo das betreffende Element gesucht werden sollen: <b>CONTENT</b> Suche im Inhalt des aktuellen Elementes (ein Unterelement) <b>CHILD</b> Sucht ein direktes Unterelement (Kindelement). <b>PARENT</b> Sucht das direkte Kontextelement (Elternelement). <b>DOC</b> Suche und Bearbeitung aller Elemente im gesamten Dokument.
<i>Anzahl</i>	Gibt an, wie viele Elemente gesucht werden sollen:

	<b>ONE</b>	Suche und Bearbeitung eines <i>Unterelementes</i> (erster Treffer),
	<b>ALL</b>	Suche und Bearbeitung aller <i>Unterelemente</i> , Standardeinstellung.
<i>Ersetzungstyp</i>		Gibt den Ersetzungstyp für die Daten an. Es stehen folgende Varianten zur Auswahl:
	<b>FULL</b>	Bearbeitet alle <i>Unterstrukturen</i> gemäß dem Skript.
	<b>ALTFULL</b>	Wie FULL, aber Entitäten werden alternativ umgesetzt.
	<b>ASCFULL</b>	Wie FULL, aber Entitäten werden nach ASCII umgesetzt.
	<b>DATA</b>	Setzt nur die Daten um. Die Elementregeln des angegebenen Elementes und aller Unterelemente werden ausgelassen.
	<b>ALT</b>	Wie DATA, aber Entitäten werden alternativ umgesetzt.
	<b>ASC</b>	Wie DATA, aber Entitäten werden nach ASCII umgesetzt.
	<b>DROP</b>	Alle <i>Unterelemente</i> und Daten werden ausgelassen.
<i>Kontextname</i>		Ein <i>Bezeichner</i> , der zur Identifizierung einer bestimmten Unterfunktion (COPY oder CUT) durch die Bedingung SUB verwendet wird. Die Bedingung prüft, ob der angegebene Kontextname mit dem Kontextnamen der aufrufenden Funktion übereinstimmt. Wenn in der aufrufenden Funktion kein Kontextbezeichner angegeben wurde, so wird der Kontextname mit dem <i>Elementnamen</i> , in dem sich die aufrufende Funktion befindet, verglichen.
<i>Trenntext</i>		Eine <i>Zeichenkette</i> , die als Abtrennung zwischen die einzelnen gefundenen Elemente eingefügt wird.
		Der COPY/CUT-Befehl führt die Elementregeln des angegebenen Elementes bei den Ersetzungstypen <b>DATA</b> , <b>ALT</b> und <b>ASC</b> nicht aus. In allen anderen Fällen werden die Elementregeln des angegebenen Elementes ausgeführt.

### Syntaxbeispiele

```
BEG=COPY (eintrag, ALL, FULL)
```

```
BEG=COPY (eintrag[typ="alpha"], ALL, FULL)
```

```
BEG=COPY (eintrag[typ="alpha"] IN liste, ALL, FULL)
```

```
BEG=COPY (index NOTIN titel, ALL, ASC)
```

```
BEG=COPY (titel, ONE, FULL)
BEG='<h1 titel="' + COPY (, DATA) + '"'>
BEG=COPY (index, ALL, DATA, index, ", ")
```

## Typ

### Anweisung

### Beispiel 1

#### Daten:

```
<dokument>
<titel>Vorwort<fn>Bitte beachten!</fn></titel>
<p>Das ist wichtig<fn>Wirklich</fn></p>
</dokument>
```

#### Skript:

```
ELEMENT dokument
BEG="<html><body>"
END="<hr>" + COPY (fn, ALL, FULL)
END="</body></html>"
ENDE

ELEMENT fn
IF SUB // durch COPY aufgerufen?
  BEG="<div>* "
  END="</div>"
ELSE
  BEG="*" // ohne COPY
  DAT=DROP
ENDIF
ENDE

ELEMENT titel
BEG="<h1>"
END="</h1>"
ENDE

ELEMENT p
BEG="<p>"
```

```
END="</p>"
```

```
ENDE
```

**Ergebnis:**

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

```
<h1>Vorwort*</h1>
```

```
<p>Das ist wichtig*</p>
```

```
<hr>
```

```
<div>* Bitte beachten!</div>
```

```
<div>* Wirklich</div>
```

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

Dieses Beispiel kopiert alle Fußnoten (Elemente `fn`) an das Ende des Dokumentes.

**Beispiel 2**

```
ELEMENT verweis
```

```
ATTLIST [refid="" ]
```

```
BEG=COPY(kapitel[id=refid], DOC, FULL)
```

```
ENDE
```

Das Element `verweis` ruft das Element `kapitel` auf, dessen Attribut `id` mir dem Attribut `refid` des Verweises übereinstimmt (klassischer Querverweis).

**Siehe auch**

BEG, CHR, COUNT, CUT, DEL, END, FILE, IMPLIED, TOKEN, VALUE, *Element-Konvertierungsregel*, *Zeichenkette*

**Benötigt werden ggfs.**

ALTERNATE, ASCII

## COUNT

---

Zählt das Vorkommen eines Elementes bis zur aktuellen Position. Wenn kein Element angegeben wurde oder das angegebene Element das eigene Element (siehe ELEMENT) ist, wird das eigene Element gezählt. Ansonsten wird das angegebene Element im erweiterten Kontext gesucht. Ist kein Suchbereich angegeben, werden alle Vorkommen dieses Elementes auf der Ebene des gefundenen Elementes gezählt. Der Ergebniswert wird an der aktuellen Position eingefügt.

### Syntax

**COUNT**{*Elementname*}?{*Suchbereich*}?{*Format*}?

*Elementname* optionaler *Bezeichner* für ein *Element*. Wenn kein Element angegeben wurde oder das angegebene Element das eigene Element (siehe ELEMENT) ist, wird das eigene Element gezählt. Ansonsten wird das angegebene Element im erweiterten Kontext gesucht.

Vorgabe ist das aktuelle Element

Suchbereich

*Suchbereich*:

**LEVEL** Sucht und zählt die Elemente auf der Ebene des gefundenen Elementes bis zur Kontextposition des angegebenen Elementes.

**DOC** Sucht und zählt alle Elemente im Dokument bis zum angegebenen Element.

Vorgabe ist **LEVEL**.

*Format*

Art der Zählung:

**NUM** numerische Ausgabe  
(1, 2, 3, 4, ...) des Zählwertes,

**ALPHA** alphanumerische Ausgabe in Kleinbuchstaben  
(a, b, c, d, ...) des Zählwertes,

**UPALPHA** alphanumerische Ausgabe in Großbuchstaben  
(A, B, C, D, ...) des Zählwertes,

**ROM** römische Ausgabe in Kleinbuchstaben  
(i, ii, iii, iv, ...) des Zählwertes,

**UPROM** römische Ausgabe in Großbuchstaben  
(I, II, III, IV, ...) des Zählwertes,

Vorgabe ist numerisch.

Das Ergebnis wird auf eine Größe von 32 Zeichen beschränkt. Größere Werte werden nicht dargestellt. Eine Fehlermeldung warnt in diesem Fall.

Alphanumerische Listen werden wie folgt gezählt:

a-z, aa-zz, aaa-zzz, ...

Römische Zahlen über 1000 erhalten pro zusätzlichen Tausender ein M.

### Syntaxbeispiele

```
BEG=COUNT() // aktuelles Element zaehlen
BEG=COUNT(,ROM) // roemisch zaehlen
BEG=COUNT(,ALPHA) // alphanumerisch zaehlen
BEG=COUNT(eintrag) // Element eintrag zaehlen
BEG=COUNT(eintrag,ALPHA)
BEG=COUNT(eintrag,ROM)
BEG=COUNT(eintrag,LEVEL,ROM)
BEG=COUNT(fn,DOC)
BEG=COUNT(kapitel,DOC,ALPHA)
```

### Typ

#### Anweisung

### Beispiel 1

#### Daten:

```
<liste>
  <eintrag>Meer</eintrag>
  <eintrag>Wasser</eintrag>
  <eintrag>Fische</eintrag>
</liste>
```

#### Skript:

```
ELEMENT eintrag
BEG="<p>"+COUNT(eintrag)+" "
END="</p>"
ENDE
```

#### Ergebnis:

```
<p>1. Meer</p>
<p>2. Wasser</p>
<p>3. Fische</p>
```

In diesem Beispiel werden die Listenelemente `eintrag` innerhalb der Liste `liste` durchnummeriert.

### Beispiel 2

#### Daten:

```
<liste typ="alpha">
  <eintrag>Meer</eintrag>
  <eintrag>Wasser</eintrag>
  <eintrag>Fische</eintrag>
</liste>
```

#### Skript:

```
ELEMENT liste
ATTLIST [typ="num"]
ENDE

ELEMENT eintrag
BEG="<p>"
IF IN liste[typ="alpha"]
  BEG=COUNT(eintrag,ALPHA)+" "
ENDIF
END="</p>"
ENDE
```

#### Ergebnis:

```
<p>a) Meer</p>
<p>b) Wasser</p>
<p>c) Fische</p>
```

In diesem Beispiel werden die Listenelemente `eintrag` innerhalb der Liste `liste` durchnummeriert und die Zählung als Buchstaben (a, b, ...) ausgegeben, wenn die Liste vom Typ `alpha` ist.

### Siehe auch

BEG, CHR, COPY, CUT, DEL, END, FILE, IMPLIED, TOKEN, VALUE, *Element-Konvertierungsregel*, *Zeichenkette*

## CUT

---

Sucht das erste Auftreten des angegebenen Elementes in der *Unterstruktur* des aktuellen Elementes und gibt die Daten anhand der angegebenen Ersetzungsregeln zurück. Die Elemente und deren Inhalte werden nach dem Kopieren vollständig gelöscht!

### Syntax 1

**CUT**(*Elementname*{*Suchbereich*}?{*Anzahl*}?, *Ersetzungstyp*  
{*Kontextname*}?{*Trenntext*}?)

### Syntax 2

**CUT**(*Elementname*{*Attributbedingung*}?{*Kontextbedingung*}?{  
*Suchbereich*}?{*Anzahl*}?, *Ersetzungstyp* {*Kontextname*}?{*Trenntext*}?)

<i>Elementname</i>	<i>Bezeichner</i> für ein <i>Element</i> .
<i>Attributbedingung</i>	<b>[Attributname Operator Zeichenkette]</b> <i>Bedingung</i> , die einen Wert eines <i>Attributs</i> abfragt. Syntax hierfür siehe <i>Attributbedingung</i>
<i>Kontextbedingung</i>	eine <i>IN</i> , <i>INX</i> oder deren <i>Negation</i> mit dem <i>NOT</i> - Präfix.
<i>Suchbereich</i>	Optionale Angabe des <i>Suchbereiches</i> . Dieser gibt an, wo das betreffende <i>Element</i> gesucht werden sollen: <b>CONTENT</b> Suche im Inhalt des aktuellen Elementes (ein <i>Unterelement</i> ) <b>CHILD</b> Sucht ein direktes <i>Unterelement</i> ( <i>Kindelement</i> ). <b>DOC</b> Suche und Bearbeitung aller <i>Elemente</i> im gesamten <i>Dokument</i> .
<i>Anzahl</i>	Gibt an, wie viele <i>Elemente</i> gesucht werden sollen: <b>ONE</b> Suche und Bearbeitung eines <i>Unterelementes</i> (erster <i>Treffer</i> ), <b>ALL</b> Suche und Bearbeitung aller <i>Unterelemente</i> , <i>Standardeinstellung</i> .
<i>Ersetzungstyp</i>	Gibt den <i>Ersetzungstyp</i> für die <i>Daten</i> an. Es stehen folgende <i>Varianten</i> zur Auswahl: <b>FULL</b> Bearbeitet alle <i>Unterstrukturen</i> gemäß dem <i>Skript</i> . <b>ALTFULL</b> Wie <b>FULL</b> , aber <i>Entitäten</i> werden alternativ umgesetzt.

- ASCFULL** Wie FULL, aber Entitäten werden nach ASCII umgesetzt.
- DATA** Setzt nur die Daten um. Die Elementregeln des angegebenen Elementes und aller Unterelemente werden ausgelassen.
- ALT** Wie DATA, aber Entitäten werden alternativ umgesetzt.
- ASC** Wie DATA, aber Entitäten werden nach ASCII umgesetzt.
- DROP** Alle *Unterelemente* und Daten werden ausgelassen.

Die Elemente und deren Inhalte werden nach dem Kopieren vollständig gelöscht!

#### *Kontextname*

Ein *Bezeichner*, der zur Identifizierung einer bestimmten Unterfunktion (COPY oder CUT) durch die Bedingung SUB verwendet wird. Die Bedingung prüft, ob der angegebene Kontextname mit dem Kontextnamen der aufrufenden Funktion übereinstimmt. Wenn in der aufrufenden Funktion kein Kontextbezeichner angegeben wurde, so wird der Kontextname mit dem *Elementnamen*, in dem sich die aufrufende Funktion befindet, verglichen.

#### *Trenntext*

Eine *Zeichenkette*, die als Abtrennung zwischen die einzelnen gefundenen Elemente eingefügt wird.

Der COPY/CUT-Befehl führt die Elementregeln des angegebenen Elementes bei den Ersetzungstypen **DATA**, **ALT** und **ASC** nicht aus.

Im Gegensatz zum COPY-Befehl werden die Daten aber aus der Quelle ausgeschnitten und sind für folgende Prozesse nicht mehr verfügbar.

#### Syntaxbeispiele

```
BEG=CUT (fn, ALL, FULL)
BEG=CUT (fn, ALL, FULL, "Fussnote")
BEG=CUT (fn, DOC, FULL)
BEG=CUT (fn, DOC, ASC)
BEG=CUT (fn, DOC, ASC, "Fussnote")
```

#### Typ

#### Anweisung

#### Beispiel

```
ELEMENT dokument
```

```
// ...  
BEG='<meta name="keywords" content="'  
BEG=CUT(index, ALL, FULL)  
BEG=' ">'  
// ...  
ENDE
```

Das Beispiel kopiert alle Stichworte (Elemente `index`) in die Metainformation, wobei diese Elemente aus den Daten "herausgeschnitten" werden.

### Siehe auch

*Zeichenkette*, BEG, CHR, COPY, COUNT, DEL, END, FILE, IMPLIED, TOKEN, VALUE, *Element-Konvertierungsregel*

## DAT

---

Steuert die Art der Datenkonvertierung für den Inhalt des betreffenden *Elements* und aller seine *Unterelemente*.

### Syntax

**DAT** = Ersetzungstyp

#### Ersetzungstyp

Gibt den Ersetzungstyp für die Daten an. Es stehen folgende Varianten zur Auswahl:

- FULL** Bearbeitet alle *Unterstrukturen* gemäß dem Skript.
- DATA** Setzt nur die Daten um, Elementregeln werden ausgelassen.
- ALTFULL** Wie FULL, aber Entitäten werden alternativ umgesetzt.
- ALT** Wie DATA, aber Entitäten werden alternativ umgesetzt.
- ASCFULL** Wie FULL, aber Entitäten werden nach ASCII umgesetzt.
- ASC** Wie DATA, aber Entitäten werden nach ASCII umgesetzt,
- DROP** Alle *Unterelemente* und Daten werden ausgelassen.

Wenn ein *Element* als Auslassung definiert wird, so wird bei der sequentiellen Konvertierung des Dokumentes dieses *Element* ausgelassen. Alle Funktionen, die als Unterfunktion Elemente aufrufen, können trotzdem diese Daten lesen und bearbeiten. Das trifft auch für die Funktionen ALTONE, ALTALL, ASCONE, ASCALL, COPY und CUT zu.

### Syntaxbeispiele

```
// Beispiel-Skript:  
ELEMENT index  
DAT=DROP  
ENDE  
// BeispielSkript:  
IF IN eintrag  
    DAT=DROP // Datenauslassen  
ENDIF
```

### Typ

#### Anweisung

### Beispiel 1

#### Daten:

```
<p>Ein Text<fn>mit Fu&uuml;note</fn> und folgender  
Text</p>
```

#### Skript:

```
ELEMENT fn
```

```
DAT=DROP
```

```
ENDE
```

```
ELEMENT p
```

```
BEG="<div>"
```

```
BEG="</div>"
```

```
ENDE
```

#### Ausgabe:

```
<div>Ein Text und folgender Text</div>
```

Alle Daten im Element `fn` werden ausgelassen.

### Beispiel 2

#### Daten:

```
<dokument>
```

```
<titel>Eine &Uuml;berschrift<fn>mit Fu&uuml;note</fn></  
titel>
```

```
<p>Ein Text<fn>mit Fu&uuml;note</fn> und folgender  
Text</p>
```

```
</dokument>
```

#### Skript:

```
ELEMENT fn
```

```
IF IN titel
```

```
DAT=DROP
```

```
BEG="*"
```

```
ELSE
```

```
BEG="<span>"
```

```
END="</span>"
```

```
ENDIF
```

```
ENDE
```

```
ELEMENT titel
```

```
BEG="<h1>"
```

```
BEG="</h1>"  
ENDE
```

```
ELEMENT p  
BEG="<div>"  
BEG="</div>"  
ENDE
```

```
ENTITY Uuml="ü"
```

### Ausgabe:

```
<h1>Eine Überschrift*</titel>  
<div>Ein Text<span>mit Fu&uuml;note</span> und folgender  
Text</div>
```

Im Kontext `titel` werden alle Daten des Elementes `fn` ausgelassen. Stattdessen wird ein `*` ausgegeben. In allen anderen Kontexten werden die Elemente `fn` konvertiert.

### Siehe auch

BEG, CHR, COPY, COUNT, DEL, END, FILE, TOKEN, VALUE, *Element-Konvertierungsregel*, *Zeichenkette*

## DEL

---

Sucht das erste Auftreten des angegebenen Elementes in der *Unterstruktur* des aktuellen Elementes und entfernt die Daten. An der aktuellen Position werden keine Daten eingefügt.

### Syntax 1

**DEL**(*Elementname*, *Suchtyp*, *Löschttyp*)

### Syntax 2

**DEL**(*Elementname*{*Attributbedingung*}{, *Suchbereich*}{, *Anzahl*}{, *Löschttyp*)

### Syntax 3

**DEL**(*Elementname*{*Attributbedingung*}{*Kontextbedingung*}{, *Suchbereich*}{, *Anzahl*}{, *Löschttyp*)

<i>Elementname</i>	<i>Bezeichner</i> für ein <i>Element</i> .
<i>Attributbedingung</i>	<b>[Attributname Operator Zeichenkette]</b> <i>Bedingung</i> , die einen Wert eines <i>Attributs</i> abfragt. Syntax hierfür siehe <i>Attributbedingung</i>
<i>Kontextbedingung</i>	eine IN, INX oder deren Negation mit dem NOT-Präfix.
<i>Suchbereich</i>	Optionale Angabe des Suchbereiches. Dieser gibt an, wo das betreffende Element gesucht werden sollen: <b>CONTENT</b> Suche im Inhalt des aktuellen Elementes (ein Unterelement) <b>CHILD</b> Sucht ein direktes Unterelement (Kindelement). <b>DOC</b> Suche und Bearbeitung aller Elemente im gesamten Dokument.
<i>Anzahl</i>	Gibt an, wie viele Elemente gesucht werden sollen: <b>ONE</b> Suche und Bearbeitung eines <i>Unterelementes</i> (erster Treffer), <b>ALL</b> Suche und Bearbeitung aller <i>Unterelemente</i> , Standardeinstellung.
<i>Löschttyp</i>	Gibt den Löschtyp für die Daten an. Es stehen folgende Varianten zur Auswahl: <b>FULL</b> Löscht alle Daten und <i>Unterstrukturen</i> .

**DATA** Löscht nur die Daten, Strukturen bleiben erhalten.

### Syntaxbeispiele

```
BEG=DEL (index, DOC, FULL)
```

```
BEG=DEL (index, DOC, DATA)
```

```
END=DEL (eintrag[typ="alpha"] IN liste, DOC, FULL)
```

### Typ

#### Anweisung

#### Beispiel

```
ELEMENT dokument
```

```
// ...
```

```
BEG=DEL (index, ALL, FULL)
```

```
// ...
```

```
ENDE
```

Das Beispiel entfernt alle Stichworte (Elemente `index`) aus dem Dokument. Bitte beachten Sie, dass bei diesem Ausgabebefehl keine Ausgabe erfolgt. Weiterhin sind diese Elemente auch für weitere Kontextanfragen nicht mehr verfügbar!

#### Siehe auch

BEG, CHR, COPY, COUNT, CUT, END, FILE, IMPLIED, TOKEN, VALUE, *Element-Konvertierungsregel*, *Zeichenkette*

## ELEMENT

---

Definiert den Aufbau und die Konvertierung des angegebenen Elementes.

### Syntax

**ELEMENT** *Elementname*

{MODEL}?

{ATTLIST}\*

{*Element-Konvertierungsregel*}\*

### ENDE

*Elementname*                      *Bezeichner* für ein *Element*.

MODEL                              definiert die Art des Inhaltsmodells mit der MODEL-Anweisung.

ATTLIST                            definiert die Attribute und ihren Vorgabewerten mit der ATTLIST-Anweisung.

*Element-Konvertierungsregel* Alle für dieses *Element* verwendeten Konvertierungsregeln. Diese können aus einer beliebigen Zusammenstellung von folgenden Komponenten bestehen:

Verzweigung                      Bedingte Verzweigungen mit IF, ELSE, ELSEIF und ENDIF. Diese Verzweigungen können beliebig tief verschachtelt werden.

Ausgabefunktion                 Mit den Ausgabefunktionen BEG (Ausgabe) und END (Ausgabe) werden für den Start und Endtag die auszugebenden Texte definiert.

Fehlermeldung                  Fehlermeldungen, die mit der ERR-Anweisung auf der Konsole ausgegeben werden sollen.

### Beispiel 1

**ELEMENT** p

```
BEG="<div>"
```

```
END="</div>"
```

ENDE

Definition der Konvertierungsregeln für das Datenelement p. Das Element wird mit dem Inhaltsmodell MIXED angenommen, da kein Datenmodell mit MODEL definiert wurde. Der Parser ruft bei jedem Auftreten des Elementes p dieses Programm (Handle) auf. Im Beispiel werden also die p-Elemente zu div-Elementen konvertiert.

### Beispiel 2

**ELEMENT** bild

```
MODEL=EMPTY
```

```
BEG="<img>"
```

```
END="</img>"
```

```
ENDE
```

Definition der Konvertierungsregeln für das Datenelement `bild`. Das Element wird mit dem Inhaltsmodell `EMPTY` definiert, d.h., in den Daten existiert kein Endtag für dieses Element. Der Parser ruft bei jedem Auftreten des Elementes `bild` dieses Programm (Handle) auf. Im Beispiel werden also die `bild`-Elemente zu `img`-Elementen konvertiert.

### Beispiel 3

**ELEMENT** bild

```
MODEL=EMPTY
```

```
ATTLIST [datei=""]
```

```
BEG="<img"
```

```
IF [datei<>""]
```

```
    BEG=' src="'+VALUE([datei])+'''
```

```
ELSE
```

```
    ERR="Dieses Bild enthält keine Dateiangabe!"
```

```
ENDIF
```

```
BEG=">"
```

```
END="</img>"
```

```
ENDE
```

Zusätzlich zum Beispiel 2 wird das Attribut `datei` definiert. Der Wert des Attributes `datei` wird als Quelle für das Element `img` in das Attribut `src` eingetragen.

### Siehe auch

*Skript, Element-Handler*

## ELSE

---

Alternativer Zweig in einer *Verzweigung*.

Dieser Zweig wird dann ausgeführt, wenn alle vorherigen Bedingungen (IF und ELSEIF) falsch waren.

### Syntax

IF *Bedingungsausdruck*

*Element-Konvertierungsregel*

{ELSEIF *Bedingungsausdruck*

*Element-Konvertierungsregel*}\*

{**ELSE**

*Element-Konvertierungsregel*}?

ENDIF

### Beschreibung

Siehe *Verzweigung*.

### Siehe auch

IF, ELSEIF, ENDIF, *Bedingung*, *Bedingungsausdruck*, *Element-Konvertierungsregel*, *Verzweigung*

## ELSEIF

---

Alternativer Zweig in einer *Verzweigung*. Der alternative Zweig wird dann geprüft, wenn der die primäre Bedingung (IF) falsch war.

Es können mehrere alternative Zweige (ELSEIF) innerhalb einer Verzweigung existieren.

Wenn die angegebene Bedingung wahr ist, wird der zugehörige Zweig ausgeführt.

### Syntax

IF *Bedingungsausdruck*

*Element-Konvertierungsregel*

{**ELSEIF** *Bedingungsausdruck*

*Element-Konvertierungsregel*}\*

{ELSE

*Element-Konvertierungsregel*}?

ENDIF

### Beschreibung

Siehe *Verzweigung*.

### Siehe auch

IF, ELSE, ENDIF, *Bedingung*, *Bedingungsausdruck*, *Element-Konvertierungsregel*, *Verzweigung*

## EMPTY

---

Prüft, ob das aktuelle *Element* leer ist.

### Syntax

#### EMPTY

### Hinweis

Das Element ist nicht leer, wenn es Unterelemente oder Zeichendaten enthält.

### Hinweis

Als Zeichendaten werden auch Whitespace (siehe Leerraum) gerechnet, wenn im Inhaltsmodell des aktuellen Elementes #PCDATA (siehe PCDATA) zugelassen ist. Mit der Befehl MODEL können Sie innerhalb des Skriptes das Inhaltsmodell des betreffenden Elementes definieren.

### Syntaxbeispiele

```
IF EMPTY
IF NOTEMPTY
```

### Typ

#### *Bedingung*

### Beispiel

#### Daten:

```
<table>
<row><entry>Zelle 1</entry><entry>Zelle 2</entry></row>
<row><entry>Zelle 3</entry><entry></entry></row>
</table>
```

#### Skript:

```
ELEMENT entry
BEG="<td>"
IF EMPTY
  BEG="&nbsp;"
ENDIF
BEG="</td>"
ENDE

ELEMENT row
```

```
BEG="<tr>"
END="</tr>"
ENDE
```

```
ELEMENT table
BEG="<table>"
END="</table>"
ENDE
```

**Ausgabe:**

```
<table>
<tr><td>Zelle 1</td><td>Zelle 2</td></tr>
<tr><td>Zelle 3</td><td>&nbsp;</td></tr>
</table>
```

**Skript:**

In diesem Beispiel werden leere Tabellenzellen (Element `entry`) in der Ausgabe (Element `td`) mit einem festen Leerzeichen (`&nbsp;`) versehen (siehe 4. Zelle).

**Siehe auch**

AFTER, BEFORE, BEG, END, FIND, FIRST, IMPLIED, IN, INX, LAST, SUB, TOPLEVEL, *Bedingungsausdruck*

## END

---

Verfügbar als Anweisung (siehe BEG) und als Bedingung (siehe BEG).

### END-Anweisung

---

Als *Anweisung*:

Definiert den Ersetzungstext für den Endtag. Bei Mehrfachdefinitionen werden diese in Reihenfolge der Ausführung im Skript umgesetzt.

Der Ersetzungstext kann durch eine beliebige Kombination von *Zeichenketten* und Makrofunktionen definiert werden, wobei diese mit dem "+"-Operator zu verbinden sind.

### Syntax

**END** = *Ausgabeanweisung* {+ *Ausgabeanweisung*}\*

*Ausgabeanweisung*     Eine *Element-Ausgabeanweisung*. Anweisung, deren Ergebnis an dieser Position ausgegeben wird.

**+**                     Verkettungsoperator. Mit diesem Operator werden Texte oder *Element-Ausgabeanweisungen* zusammengeführt. Die Ausgabe erfolgt in Reihenfolge der Verkettung von links nach rechts.

### Syntaxbeispiele

```
END='<p>Beispiel</p>'  
END='<p class="hinweis">Hinweis</p>'  
END='<p>Beispiel</p>'+CHR(13)+"<p>...</p>"  
END='<p>Beispiel: '+VALUE([nr])+"</p>"  
// typisches Elementumsetzung:  
BEG="<p>"  
END="</p>"
```

### Typ

*Anweisung*

### Beispiel

```
ELEMENT p  
BEG="<div>"  
END="</div>"  
ENDE
```

Mit der Anweisung END wird in diesem Beispiel statt dem Endtag </p> ein </div> ausgegeben.

**Siehe auch**

BEG, CHR, COPY, COUNT, CUT, DEL, ERR, FILE, IMPLIED, TOKEN, VALUE, *Element-Ausgabeeweisung*, *Element-Konvertierungsregel*, *Verzweigung*, *Zeichenkette*

## END-Bedingung

---

Prüft, ob gerade ein Endtag bearbeitet wird.

### Syntax

#### END

Die Bedingung END liefert *Wahr* bei der Bearbeitung eines Endtags. Bei der Bearbeitung eines Starttags wird *Falsch* zurückgegeben. Mit der Bedingung END (Abfrage) können Verzweigungen so gesteuert werden, so dass diese nur beim Auftreten eines Endtags ausgeführt werden.

### Hinweis

Da die Konvertierungsregeln für komplette Elemente definiert werden, werden alle Kontextabfragen sowohl für den Starttag als auch für den Endtag durchgeführt. Zur Laufzeit-Optimierung des Skriptes kann es erforderlich sein, diese Kontextabfragen jeweils nur für den Start- oder Endtag zuzulassen.

### Syntaxbeispiele

```
// Beispiel-Block:
IF END
  // Dieser Block wird nur fuer den Endttag
  // ausgefuehrt
  END="</p>"
  IF IN eintrag LAST
    END="<hr>"
  ENDIF
  BEG="--: (" // Dieser Befehl wird nie ausgefuehrt!
ENDIF
```

### Typ

#### *Bedingung*

### Beispiel

#### Daten:

```
<kapitel id="TBB.4"><titel>Berechnung</titel>
```

#### Skript:

```
ELEMENT titel
  IF END
    END="</h1>"
```

```
ERR="Ende des Titels"  
ELSE  
  BEG="<h1>"  
  IF kapitel[id<>"]  
    BEG='<a name="' + VALUE(kapitel[id]) + "'></a>'  
  ENDIF  
ENDIF  
ENDE
```

**Ergebnis:**

```
<h1><a name="TBB.4"></a>Berechnung</h1>
```

Die Abfrage nach dem *Attribut* `id` wird in diesem Fall nur durchgeführt, wenn ein Starttag konvertiert wird. Im Ergebnis der Ausgabe ergeben sich keine Unterschiede, allerdings in der verbesserten Laufzeit des Programms. Zusätzlich wird beim Erreichen des Endtags eine Benutzerfehlermeldung ausgegeben.

**Siehe auch**

AFTER, BEFORE, BEG, EMPTY, FIND, FIRST, IMPLIED, IN, INX, LAST, SUB, TOPLEVEL, *Bedingungsausdruck*

**ENDE**

Ende einer Elementdefinition (ELEMENT).

**Syntax**

ELEMENT *Elementname*

{MODEL}?

{ATTLIST}?

{*Element-Konvertierungsregel*}\*

**ENDE****Beschreibung**

Siehe ELEMENT.

**Siehe auch**

ELEMENT, *Skript*, *Element-Handler*

**ENDIF**

---

Ende der *Verzweigung*.

**Syntax**

IF *Bedingungsausdruck*

*Element-Konvertierungsregel*

{ELSEIF *Bedingungsausdruck*

*Element-Konvertierungsregel*}\*

{ELSE

*Element-Konvertierungsregel*}?

**ENDIF****Beschreibung**

Siehe *Verzweigung*.

**Siehe auch**

IF, ELSE, ELSEIF, *Bedingung*, *Bedingungsausdruck*, *Element-Konvertierungsregel*, *Verzweigung*

## ENTITY

---

Definiert die Entitäten-Ersetzung.

### Syntax 1

**ENTITY** *Name* = *Ausgabeanweisung* {+ *Ausgabeanweisung* }\* { *Alternativ-Ausgabe*? } { *Ascii-Ausgabe*? }

### Syntax 2

**ENTITY** #*Num* = *Ausgabeanweisung* {+ *Ausgabeanweisung* }\* { *Alternativ-Ausgabe*? } { *Ascii-Ausgabe*? }

### Syntax 3

**ENTITY** #*xHex* = *Ausgabeanweisung* {+ *Ausgabeanweisung* }\* { *Alternativ-Ausgabe*? } { *Ascii-Ausgabe*? }

### Syntax 3 (nur SGML)

**ENTITY** #*Funktion* = *Ausgabeanweisung* {+ *Ausgabeanweisung* }\* { *Alternativ-Ausgabe*? } { *Ascii-Ausgabe*? }

<i>Name</i>	gültiger XML-/SGML-Bezeichner für eine Entität. Das Entitäten-Startzeichen (reference open) "&" und das Entitäten-Endezeichen (reference close) ";" sind wegzulassen.
<i>Num</i>	Dezimaler Wert eines Zeichens. Folgende Zeichen sind zulässig: 0123456789 Der Wert muss größer als 0 sein. Der Wert darf 1114111 nicht überschreiten.
<i>Hex</i>	Hexadezimaler Wert eines Zeichens. Folgende Zeichen sind zulässig: 0123456789ABCDEF Der Wert muss größer als 0 sein. Die Groß/Kleinschreibung wird ignoriert. Der Wert muss größer als 0 sein. Der Wert darf x10FFFF nicht überschreiten.
<i>Funktion</i>	gültiger XML-/SGML-Bezeichner für eine Funktionszeichenreferenz. Achtung! Es dürfen nur

die folgenden vordefinierten Bezeichner verwendet werden:

RE	13
RS	10
TAB	9
SPACE	31

Die Funktionszeichen werden als „numerische“ Werte gewertet. Es darf deshalb nur eine Definition eines Zeichen im Skript geben (z.B. entweder #9 oder #TAB, aber nie gleichzeitig).

Das Entitäten-Startzeichen (reference open) "&" und das Entitäten-Endezeichen (reference close) ";" sind wegzulassen.

*Ausgabeeweisung* Eine *Entitäten-Ausgabeeweisung*. Anweisung, deren Ergebnis an dieser Position ausgegeben wird.

**+** Verkettungsoperator. Mit diesem Operator werden Texte oder *Entitäten-Ausgabeeweisungen* zusammengeführt. Die Ausgabe erfolgt in Reihenfolge der Verkettung von links nach rechts.

*Alternativ-Ausgabe* Definiert eine alternative Entitätenkonvertierung mit dem Befehl ALTERNATE. Die alternative Konvertierung wird mit dem Befehl COPY und dem Ersetzungstyp **ALT** aufgerufen.

*Ascii-Ausgabe* Definiert eine alternative ASCII-Entitätenkonvertierung mit dem Befehl ASCII. Die alternative ASCII-Konvertierung wird mit dem Befehl COPY und dem Ersetzungstyp **ASC** aufgerufen.

### Hinweis

Rekursionsschutz! Bei Verwendung von VALUE innerhalb einer *Entitäten-Ausgabeeweisung* werden nur Textbestandteile (CDATA) des betreffenden *Attributwerts* zurückgegeben und wiederum enthaltene Entitäten ausgelassen!

### Syntaxbeispiele

```
ENTITY auml="&auml;" ALTERNATE="ae"  
ENTITY auml="&uuml;" ALTERNATE="ue" ASCII="ü"  
ENTITY #0214="&Ouml;"  
ENTITY #x00DC="&Uuml;"  
ENTITY #TAB=" " // Tabulator
```

## TYP

### Anweisung

#### Beispiel 1

**ENTITY** auml="ä"

**ENTITY** Auml="Ä"

**ENTITY** ouml="ö"

**ENTITY** Ouml="Ö"

**ENTITY** uuml="ü"

**ENTITY** Uuml="Ü"

**ENTITY** szlig="ß"

In diesem Beispiel werden die deutschen Umlaute und das "ß"-Zeichen aus den ISO-Entitäten in ANSI-Zeichen konvertiert.

#### Beispiel 2

**ENTITY** auml="&auml;" ALTERNATE="ae" ASCII="ä"

**ENTITY** Auml="&Auml;" ALTERNATE="Ae" ASCII="Ä"

**ENTITY** ouml="&ouml;" ALTERNATE="oe" ASCII="ö"

**ENTITY** Ouml="&Ouml;" ALTERNATE="Oe" ASCII="Ö"

**ENTITY** uuml="&uuml;" ALTERNATE="ue" ASCII="ü"

**ENTITY** Uuml="&Uuml;" ALTERNATE="Ue" ASCII="Ü"

**ENTITY** szlig="&szlig;" ALTERNATE="sz" ASCII="ß"

In diesem Beispiel werden die deutschen Umlaute und das "ß"-Zeichen in der Standardkonvertierung 1:1 (ebenfalls als Entitäten) ausgegeben. Alle Datenbereiche, die mit dem DAT-Befehl DAT=ALT definiert sind, werden mit den Ersetzungszeichen von ALTERNATE konvertiert. Alle Datenbereiche, die mit dem DAT-Befehl DAT=ASC definiert sind, werden mit den Ersetzungszeichen von ASCII konvertiert.

#### Beispiel 3

**ENTITY** #228="&auml;"

**ENTITY** #xC4="&Auml;"

**ENTITY** #TAB=" "

In diesem Beispiel wird der numerische Zeichenverweis 228 (Angabe dezimal) in die HTML-Entität &auml; konvertiert, der numerische Zeichenverweis xC4 (Angabe hexadezimal) in die HTML-Entität &Auml;. Der Funktionszeichen #TAB wird als Leerzeichen gewertet. Es werden alle numerischen Zeichenverweise mit dem Wert 9 (Tabulator, durch #TAB definiert) ebenfalls in Leerzeichen konvertiert.



**Siehe auch**

ALTERNATE, ASCII, *Skript*

## ERR

---

Definiert den Text einer Benutzer-Fehlermeldung. Bei Mehrfachdefinitionen werden diese in Reihenfolge der Ausführung im Skript ausgegeben. Die Ausgabe des Textes erfolgt in der Standardausgabe (Konsole).

### Syntax

**ERR** = Meldung {+ Meldung}\*

Meldung                      Eine *Zeichenkette*, die an die Konsole ausgegeben wird.

**+**                              Verkettungsoperator. Mit diesem Operator werden Texte zusammengeführt. Die Ausgabe erfolgt in Reihenfolge der Verkettung von links nach rechts.

### Hinweis

Zusätzlich zur Ausgabe des Textes wird die Positionsangabe der aktuellen Daten (Dateiname, Zeile und Spalte) ausgegeben.

### Syntaxbeispiele

```
ERR="Fehler"

// Beispiel-Skript:
IF [typ=""]
  ERR="Kein Typ definiert!"
ENDIF
```

### Typ

#### Anweisung

#### Beispiel

```
<dokument>
<titel>Kapitel&uuml;berschrift<fn>Fu&uuml;note</fn></
titel>
<p>Text</p>
<dokument>
```

#### Skript:

```
ELEMENT fn
IF IN titel
  ERR="Achtung, Fußnote in Überschrift gefunden!"
ENDIF
```

**Konsolenausgabe:**

```
WARNUNG (Skript): im Element fn
in Datei Muster.xml in Zeile=10, Zeichen=27.
Achtung, Fußnote in Überschrift gefunden!
```

Dieses Skript erzeugt bei allen Fußnotenelementen `fn`, die sich in Überschriften `titel` befinden, eine Fehlermeldung.

**Siehe auch**

BEG, CHR, COPY, COUNT, CUT, DEL, END, FILE, TOKEN, VALUE, *Element-Ausgabeeanweisung*, *Element-Konvertierungsregel*, *Verzweigung*, *Zeichenkette*

## FILE

---

Ermittelt die Dateibezeichnung (File) einer externen NDATA-Entität (z.B. Grafik), die durch einen Attributwert referenziert wird. Der ermittelte Dateiname wird an der aktuellen Position eingefügt. Bei Angabe einer Extension im zweiten Parameter, wird diese Extension statt der Original-Dateinamenserweiterung zurückgegeben.

Mit diesem Befehl können auch beliebige andere externe Entitäten aufgelöst werden. Im Ergebnis wird SYSTEM-Identifikator der Entität zurückgegeben.

### Syntax

**FILE**([*Attributname*],[*Extension*]?)

<i>Attributname</i>	<i>Bezeichner</i> für den Namen eines <i>Attributes</i> .
<i>Extension</i>	optionale Dateinamenserweiterung, die statt der Original-Dateinamenserweiterung der Datei zurückgegeben werden soll. Die Extension muss als <i>Zeichenkette</i> angegeben werden! Bei Angabe einer leeren <i>Zeichenkette</i> , wird der Original-Dateinamenserweiterung ausgegeben.

### Syntaxbeispiele

```
BEG=''
BEG=''
```

### Typ

*Anweisung*

### Beispiel 1

Daten:

```
<?xml version="1.0"?>
<!DOCTYPE dokument PUBLIC "-//Seume//DTD Handbuch
V1.00//DE" [
<!ENTITY entity.001 SYSTEM "grafik.bmp" NDATA bmp>
]>
<dokument>
  <bild entity="entity.001"/>
</dokument>
```

Skript:

```
ELEMENT bild
ATTLIST [entity=""]
BEG=''
```

ENDE

Ergebnis:

```

```

Mit dem Befehl FILE wird der Dateiname der Grafik aus der Entitäten-deklaration ermittelt und als Attribut `src` in das HTML-Bild `img` eingetragen.

## Beispiel 2

Daten:

```
<?xml version="1.0"?>
<!DOCTYPE dokument PUBLIC "-//Seume//DTD Handbuch
V1.00//DE" [
<!ENTITY entity.001 SYSTEM "grafik.bmp" NDATA bmp>
]>
<dokument>
  <bild entity="entity.001"/>
</dokument>
```

Skript:

```
ELEMENT bild
ATTLIST [entity=""]
BEG=''
ENDE
```

Ergebnis:

```

```

Mit dem Befehl FILE wird der Dateiname der Grafik ermittelt und dessen Dateiergung auf `gif` geändert. Achtung: die Grafikdatei wird nicht umbenannt.

## Siehe auch

BEG, CHR, COPY, COUNT, CUT, DEL, END, IMPLIED, TOKEN, VALUE,  
*Element-Konvertierungsregel, Zeichenkette*

## FIND

---

Prüft, ob in der untergeordneten Struktur des Elementes das angegebene *Element* vorhanden ist. Mit der Kontextbedingung kann zusätzlich ein vordefinierter Kontext-Fall als Bedingung gesetzt werden.

### Syntax 1

**FIND**(*Elementname*)

### Syntax 2

**FIND**(*Elementname*{*Attributbedingung*})

### Syntax 3

**FIND**(*Elementname*{*Attributbedingung*}?{*Kontextbedingung*}?)

### Syntax 4

**COPY**(*Elementname*{*Attributbedingung*}?{*Kontextbedingung*}?{*Suchbereich*}?)

<i>Elementname</i>	Bezeichner für ein <i>Element</i> .
<i>Attributbedingung</i>	[ <i>Attributname Operator Zeichenkette</i> ] oder [ <i>Attributname Operator Attributname</i> ] <i>Bedingung</i> , die einen Wert eines <i>Attributs</i> abfragt. Syntax hierfür siehe <i>Attributbedingung</i>
<i>Kontextbedingung</i>	eine IN, INX oder deren Negation mit dem NOT-Präfix.
<i>Suchbereich</i>	Gibt an, ob ein <i>Unterelement</i> oder ein Element im Dokument gesucht werden sollen: <b>CONTENT</b> Suche im Inhalt des aktuellen Elementes (ein Unterelement) <b>CHILD</b> Sucht ein direktes Unterelement (Kindelement). <b>PARENT</b> Sucht das direkte Kontextelement (Elternelement). <b>DOC</b> Suche eines Elementes im gesamten Dokument.

### Syntaxbeispiele

```
IF FIND(eintrag)
```

```
IF FIND(eintrag[typ="alpha"])
IF FIND(eintrag[typ="alpha"] IN liste)
IF FIND(eintrag[typ="alpha"] NOTIN vorwort)
IF FIND(eintrag IN liste)
IF FIND(eintrag INX kapitel)
IF FIND(fn NOTIN titel)
IF FIND(eintrag NOTINX fn)
IF NOTFIND(eintrag)
IF NOTFIND(eintrag[typ="alpha"])
IF NOTFIND(eintrag[typ="alpha"] IN liste)
```

## Typ

### *Bedingung*

### Beispiel

#### Daten:

```
<dokument>
<p>Auto<fn>Erste Fu&uuml;note</fn></p>
<p>Fahrrad<fn>Zweite Fu&uuml;note</fn></p>
</dokument>
```

#### Skript:

```
ELEMENT dokument
BEG="<html><body>"
IF FIND(fn)
  END="<hr><ul>"+COPY(fn,ALL,FULL)+"</ul>"
ENDIF
END="</body></html>"
ENDE

ELEMENT fn
IF SUB
  BEG="<li>"
  END="</li>"
ELSE
  DAT=DROP
ENDIF
ENDE
```

```
ELEMENT p
BEG="<p>"
END="</p>"
ENDE
```

```
ENTITY szlig="ß"
```

**Ergebnis:**

```
<html><body>
<p>Auto</p>
<p>Fahrrad</p>
<hr><ul>
<li>Erste Funote</li>
<li>Zweite Funote</li>
</ul>
</body></html>
```

Dieses Beispiel wird mit FIND geprüft, ob Fußnoten  $f_n$  vorhanden sind. Wenn Fußnotenelemente  $f_n$  vorhanden sind, werden diese am Ende des Dokumentes als Listenelemente in eine neue Liste eingetragen. Werden keine Fußnoten gefunden, wird keine Liste erzeugt.

**Siehe auch**

AFTER, BEFORE, BEG, EMPTY, END, FIRST, IMPLIED, IN, INX, LAST, SUB, TOPLEVEL, *Bedingungsausdruck*

## FIRST

---

Prüft, ob das aktuelle *Element* das erste innerhalb des übergeordneten Elementes ist.

### Syntax

#### FIRST

### Syntaxbeispiele

```
IF FIRST
IF IN eintrag FIRST
IF NOTFIRST
IF IN eintrag NOTFIRST
```

### Typ

#### Bedingung

### Beispiel

#### Daten:

```
<dokument>
<liste>
<eintrag>Elefanten</eintrag>
<eintrag>Tiger</eintrag>
<eintrag>Affen</eintrag>
</liste>
</dokument>
```

#### Skript:

```
ELEMENT dokument
BEG="<html><body>"
END="</body></html>"
ENDE
```

```
ELEMENT liste
BEG="<ul>"
END="</ul>"
ENDE
```

```
ELEMENT eintrag
BEG="<li>"
```

```
IF FIRST
  END=" (FIRST) "
ENDIF
IF NOTFIRST
  END=" (NOTFIRST) "
ENDIF
END="</li>"
ENDE
```

**Ergebnis:**

```
<html><body>
<ul>
<li>Elefanten (FIRST)</li>
<li>Tiger (NOTFIRST)</li>
<li>Affen (NOTFIRST)</li>
</liste>
</ul>
</body></html>
```

Dieses Beispiel wird mit `FIRST` geprüft, ob der jeweilige Listeneintrag eintrag der erste Eintrag der Liste `liste` ist. In diesem Fall wird der Text "`(FIRST)`" angehängt.

**Siehe auch**

AFTER, BEFORE, BEG, EMPTY, END, FIND, IN, INX, LAST, SUB, TOP-LEVEL, *Bedingungsausdruck*

## IF

---

Leitet eine neue *Verzweigung* ein.

Wenn der *Bedingungsausdruck* wahr ist, werden die *Element-Konvertierungsregel* dieses Zweiges ausgeführt und die *Verzweigung* beendet. Wenn der *Bedingungsausdruck* falsch ist, werden die alternativen Zweige (ELSEIF und ELSE) geprüft bzw. ausgeführt.

### Syntax

**IF** *Bedingungsausdruck*

*Element-Konvertierungsregel*

{ELSEIF *Bedingungsausdruck*

*Element-Konvertierungsregel*}\*

{ELSE

*Element-Konvertierungsregel*}?

ENDIF

### Beschreibung

Siehe *Verzweigung*.

### Siehe auch

ELSE, ELSEIF, ENDIF, *Bedingung*, *Bedingungsausdruck*, *Element-Konvertierungsregel*, *Verzweigung*

## IMPLIED

---

Überprüfung oder Ausgabe eines impliziten „vererbten“ Attributwertes. Das sind Attributwerte, die entweder durch das eigene oder ein beliebiges übergeordnete Element gegeben sind. Verfügbar als Anweisung (siehe IMPLIED) und als Bedingung (siehe IMPLIED).

Bei Verwendung als Bedingung wird der Wahrheitswert zurückgeben.

Bei Verwendung als Anweisung wird der Attributwert zurückgegeben.

### IMPLIED-Anweisung

---

Ausgabe eines impliziten „vererbten“ *Attributwertes*. Sucht das angegebene Attribut im aktuellen *Element* und falls es hier nicht belegt ist oder kein Vorgabewert dafür definiert wurde, in den übergeordneten Elementen.

#### Syntax 1

**IMPLIED**(*[Attributname]*)

#### Syntax 2

**IMPLIED**(*[Attributname]*,*{Operation|Ersetzungstyp}*)

#### Syntax 3

**IMPLIED**(*Elementname**[Attributname]*)

#### Syntax 4

**IMPLIED**(*Elementname**[Attributname]*,*{Operation|Ersetzungstyp}*)

*Elementname*      Optionaler *Bezeichner* für ein *Element*. Dieses *Element* kann in einem beliebigen Kontext zu dem aktuellen *Element* stehen. Es wird der vererbte Attributwert auf dieses angegebene Element errechnet.

*Attributname*      *Bezeichner* für den Namen eines *Attributes*.

*Operation*      Optionale *Zeichenkette* für eine Umrechnungsoperation und/oder Formatierung des Attributwertes. Diese Zeichenkette muss in Anführungszeichen ("..." oder '...') angegeben werden. Folgende interne Syntax innerhalb der *Zeichenkette* wird verwendet:

*{Faktor}?{Einheit}?{:Offset}?{(Format)}?*

*Faktor*      Ein numerischer Wert, der als Korrekturfaktor für den Attributwert verwendet wird.

- Einheit** Angabe einer neuen Maßeinheit, in die der Attributwert umgerechnet wird. Folgende Maßeinheiten werden als Umrechnungseinheit unterstützt:
- pt** Punkt
  - mm** Millimeter
  - cm** Zentimeter
  - in** Inch, Zoll
- Die Einheit selbst wird im Ergebnis **nicht** ausgegeben!
- Achtung: Eine Umrechnung erfolgt nur dann, wenn der Attributwert ebenfalls mit einer Maßeinheit versehen ist!
- Offset** Ein numerischer Wert, der als Korrekturoffset für den Attributwert verwendet wird. Folgende Werte sind möglich:
- +** numerischer Wert für ein positives Offset
  - numerischer Wert für ein negatives Offset.
- Bitte achten Sie auf die Eingabe des Doppelpunktes am Beginn des Offsets!
- Format** Definition des Ausgabezahlenformates. Bitte stets die Klammern setzen, z.B. (**# . ##**). Folgende Platzhalterzeichen können verwendet werden:
- #** Platzhalter für eine Dezimalzahl
  - .** Dezimalpunkt
- Beispiele:

Beispiele	<b>0</b>	<b>12</b>	<b>5467,872</b>	<b>0,6785</b>
<b>#</b>	0	12	5468	1
<b>##</b>	0,00	12,00	5467,87	0,68
<b>###</b>	000	012	5468	001
(ohne)	0	12	5467,872	0,6785

Wenn kein Format definiert wurde, gelten folgende Voreinstellungen, wenn eines der Zielformate angegeben wurde:

<b>pt</b>	#.#
<b>mm</b>	#.##
<b>cm</b>	#.###
<b>in</b>	#.####
<b>%</b>	#.#
<b>%%</b>	#.#
-	(automatisch)

Wenn keine Operation angegeben wird, erfolgt auch keine Umrechnung und Formatierung!

Wenn kein Format angegeben wird, werden die Dezimalstellen automatisch erstellt (Ganzzahlen ohne Dezimalpunkt, ansonsten bis zu 8 Dezimalstellen).

### Ersetzungstyp

Gibt den Ersetzungstyp für die Daten an. Es stehen folgende Varianten zur Auswahl:

**DATA** Die enthaltenen Entitäten werden nach den definierten Ersetzungsregeln (siehe ENTITY) umgesetzt.

**ALT** Enthaltene Entitäten werden alternativ (siehe ALTERNATE) umgesetzt.

**ASC** Enthaltene Entitäten werden nach ASCII (siehe ASCII) umgesetzt.

Wenn kein Wert angegeben wird, wird der Ersetzungstyp **DATA** als Standard verwendet.

Wenn statt dem Ersetzungstyp eine *Operation* angegeben wird, werden die Daten ohne Entitäten-ersetzung konvertiert. Die *Operation* geht in der Regel von numerischen Daten aus!

### Syntaxbeispiele

```
BEG=IMPLIED([typ])
```

```
BEG=IMPLIED(abschnitt[typ])
```

```
BEG=IMPLIED([width],"#.##")
```

```
BEG=IMPLIED([width],"1.5:(#.##)")
```

```
BEG=IMPLIED([width],"1.5pt:+100(###)")
```

```
BEG=IMPLIED(entry[width],"#.##")
```

```
BEG=IMPLIED(entry[width],"1.5:(#.##)")
```

```
BEG=IMPLIED(entry[width],"1.5pt:+100(###)")
```

```
BEG=IMPLIED(abschnitt[kategorie],ASC)
```

```
BEG=IMPLIED(abschnitt[kategorie],ALT)
```

**Beispiel****Daten:**

```
<kapitel ausrichtung="links">
<p>Text</p>
</kapitel>
```

**Skript:**

ELEMENT p

```
BEG='<DIV CLASS="' + IMPLIED ([ausrichtung]) + '"/>'
END="</DIV>"
```

ENDE

**Ergebnis:**

```
<DIV CLASS="links"/>Text</DIV>
```

Mit der Attributabfrage (IMPLIED) wird der vererbte Wert des Attributes *ausrichtung* abgefragt und als HTML-Klasse ausgegeben.

**Siehe auch**

BEG, CHR, COPY, COUNT, CUT, DEL, END, FILE, IMPLIED, TOKEN, VALUE, *Element-Konvertierungsregel*, *Zeichenkette*,

## IMPLIED-Bedingung

---

Abfrage eines vererbten Attributwertes.

Im Gegensatz zur *Attributbedingung*, die den Attributwert des aktuellen Elementes oder des angegebenen Elementes abfragt, kann mit IMPLIED der von einer übergeordneten Struktur auf das aktuelle Element vererbte Attributwert abgefragt werden.

### Syntax 1

**IMPLIED**([*Attributname Operator Wert*])

### Syntax 2

**IMPLIED**(*Elementname*[*Attributname Operator Wert*])

*Elementname*            Optionaler *Bezeichner* für ein *Element*. Dieses *Element* kann in einem beliebigen Kontext zu dem aktuellen *Element* stehen. Die Attributabfrage wird auf dieses Element angewendet. Es wird der vererbte Attributwert auf das angegeben Element errechnet.

*Attributname*            *Bezeichner* für den Namen eines *Attributes*.

*Operator*                Vergleichsoperator:

<	kleiner
>	größer
<=	kleiner gleich
>=	größer gleich
<>	ungleich (unabhängig der Groß- und Kleinschreibung)
=	gleich (unabhängig der Groß- und Kleinschreibung)
==	identisch (mit Beachtung der Groß- und Kleinschreibung)
!=	nicht identisch (mit Beachtung der Groß und Kleinschreibung)

*Wert*                     *Zeichenkette*, für den *Attributwert*.

### Syntaxbeispiele

```
IF IMPLIED ([typ<>""])
IF IMPLIED (abschnitt [typ=="vorwort"])
IF IMPLIED (abschnitt [typ!="vorwort"])
IF IMPLIED (abschnitt [typ<>"vorwort"])
```

## Typ

### Bedingung

### Beispiel

#### Daten:

```
<eintrag ausrichtung="rechts">
  <p>100</p>
</eintrag>
```

#### Skript:

```
ELEMENT p
BEG='<p
IF IMPLIED ([ausrichtung="rechts"])
  BEG=' class="rechts" '
ENDIF
BEG=">"
END='</p>'
ENDE
```

#### Ausgabe:

```
<p class="rechts">100</p>
```

Diese Bedingung prüft, ob die Eigenschaft `ausrichtung` auf das Element `p` vererbt wurde und den Wert `rechts` besitzt. In diesem Fall wird in der Ausgabe das Attribut `class` mit dem Wert `rechts` erzeugt.

### Siehe auch

*VALUE, Attributbedingung, Attributname, Attributwert*

## IN

---

Prüft, ob das aktuelle *Element* ein direktes *Unterelement* eines der angegebenen Elemente ist.

Diese Abfrage liefert dann *Richtig*, wenn das aktuelle *Element* ein direktes *Unterelement* eines der angegebenen Elemente ist.

Durch Angabe einer *Attributbedingung* kann diese für das angegebene Element geprüft werden. Die Abfrage liefert dann *Richtig*, wenn das aktuelle Element ein direktes *Unterelement* des angegebenen Elementes ist und die *Attributbedingung* für das angegebene Element erfüllt wird.

### Syntax 1

**IN** {*Elementname*}{*Attributbedingung*}?

### Syntax 2

**IN**({*Elementname*}{*Attributbedingung*}?)

*Elementname*                      *Bezeichner* für ein *Element*.

*Attributbedingung*                *Bedingung*, die einen Wert eines *Attributs* abfragt.  
Syntax hierfür siehe *Attributbedingung*.

Diese Abfrage liefert dann *Richtig*, wenn das aktuelle *Element* ein direktes *Unterelement* des angegebenen Elementes ist.

Durch Angabe einer *Attributbedingung* kann diese für das angegebene Element geprüft werden. Die Abfrage liefert dann *Richtig*, wenn das aktuelle Element ein direktes *Unterelement* des angegebenen Elementes ist und die *Attributbedingung* für das angegebene Element erfüllt wird.

### Hinweis

Wird als Parameter einer Abfrage ein Element **ohne Klammern** angegeben und liefert diese Abfrage *Richtig*, dann wird dieses Element für die folgende Abfrage als das aktuelle Element betrachtet. Das Element des Parameters wird also zum aktuellen Element erklärt!

Wird als Parameter einer Abfrage ein Element **in Klammern** angegeben, bleibt das aktuelle Element unverändert.

### Syntaxbeispiele

```
IF IN kapitel
IF IN(kapitel)
IF IN eintrag IN liste
IF IN(eintrag) AFTER(titel)
IF AFTER titel IN eintrag
```

```
IF IN p NOTBEFORE liste
IF IN p NOTBEFORE (liste)
```

## Typ

### Bedingung

### Beispiel 1

#### Daten:

```
<liste typ="alpha">
  <eintrag>Meer</eintrag>
  <eintrag>Wasser</eintrag>
  <eintrag>Fische</eintrag>
</liste>
```

#### Skript:

```
ELEMENT liste
  ATTLIST [typ="num"]
ENDE

ELEMENT eintrag
BEG="<p>"
IF IN liste[typ="alpha"]
  BEG=COUNT(eintrag,ALPHA)+" "
ENDIF
END="</p>"
ENDE
```

#### Ergebnis:

```
<p>a) Meer</p>
<p>b) Wasser</p>
<p>c) Fische</p>
```

Die Listenelemente `eintrag` werden innerhalb der Liste `liste` durchnummeriert und die Zählung als Buchstaben (a, b, ...) ausgegeben, wenn die Liste vom Typ `alpha` ist. Die Kontextabfrage `IN` wird zur Bestimmung des Listentyps verwendet.

### Beispiel 2

#### Daten:

```
<liste>
  <eintrag>
```

```
<p>Wassergetier:</p>
<liste typ="alpha">
  <eintrag>Fische</eintrag>
  <eintrag>Quallen</eintrag>
</liste>
</eintrag>
<eintrag>
  <p>Landgetier:</p>
  <liste typ="alpha">
    <eintrag>Pferde</eintrag>
    <eintrag>Hunde</eintrag>
  </liste>
</eintrag>
</liste>
```

**Skript:**

```
ELEMENT liste
  ATTLIST [typ="num"]
ENDE

ELEMENT p
  BEG="<p>"
  BEG="</p>"
ENDE

ELEMENT eintrag
BEG="<p>"
IF IN liste[typ="alpha"]
  BEG=COUNT(eintrag,ALPHA)+") "
ELSE
  BEG=COUNT(eintrag)+". "
ENDIF
END="</p>"
ENDE
```

**Ergebnis:**

```
<p>1. Wassergetier:</p>
<p>a) Fische</p>
<p>b) Quallen</p>
```

```
<p>c) Fische</p>
<p>2. Landgetier:</p>
<p>a) Pferde</p>
<p>b) Hunde</p>
```

Die Listenelemente `eintrag` werden innerhalb der Liste `liste` durchnummeriert und die Zählung als Buchstaben (a, b, ...) für die Liste vom Typ `alpha` und die Zählung mit Ziffern (1, 2., ...) für die Liste vom Typ `num` ausgegeben. Die Kontextabfrage `IN` muss hier verwendet werden, um sicherzustellen, dass bei verschachtelten Listen der Typ der Liste jeweils korrekt bestimmt wird.

### Siehe auch

AFTER, BEFORE, BEG, EMPTY, END, FIND, FIRST, INX, LAST, SUB, TOP-LEVEL, *Bedingungsausdruck*

## INX

---

Prüft, ob das aktuelle *Element* ein beliebiges *Unterelement* des angegebenen Elementes ist.

Diese Abfrage liefert dann *Richtig*, wenn sich das aktuelle *Element* innerhalb der *Unterstruktur* des angegebenen Elementes befindet.

Durch Angabe einer *Attributbedingung* kann diese für das angegebene Element geprüft werden. Die Abfrage liefert dann *Richtig*, wenn sich das aktuelle Element in der *Unterstruktur* des angegebenen Elementes befindet und die *Attributbedingung* für das angegebene Element erfüllt wird.

### Syntax 1

**INX** {*Elementname*}{*Attributbedingung*}?

### Syntax 2

**INX**({*Elementname*}{*Attributbedingung*}?)

*Elementname*                      *Bezeichner* für ein *Element*.

*Attributbedingung*                *Bedingung*, die einen Wert eines *Attributs* abfragt.  
Syntax hierfür siehe *Attributbedingung*.

Diese Abfrage liefert dann *Richtig*, wenn sich das aktuelle *Element* innerhalb der *Unterstruktur* des angegebenen Elementes befindet.

Durch Angabe einer *Attributbedingung* kann diese für das angegebene Element geprüft werden. Die Abfrage liefert dann *Richtig*, wenn sich das aktuelle *Element* innerhalb der *Unterstruktur* des angegebenen Elementes befindet und die *Attributbedingung* für das angegebene Element erfüllt wird.

### Hinweis

Wird als Parameter einer Abfrage ein Element **ohne Klammern** angegeben und liefert diese Abfrage *Richtig*, dann wird dieses Element für die folgende Abfrage als das aktuelle Element betrachtet. Das Element des Parameters wird also zum aktuellen Element erklärt!

Wird als Parameter einer Abfrage ein Element **in Klammern** angegeben, bleibt das aktuelle Element unverändert.

### Syntaxbeispiele

```
IF INX kapitel
IF INX(kapitel)
IF INX liste
IF INX p NOTBEFORE liste
IF INX p NOTBEFORE(liste)
```

## Typ

### Bedingung

### Beispiel

#### Daten:

```
<table>
<row><entry>Zelle 1</entry><entry>
<liste>
<eintrag>Maus</eintrag>
<eintrag>Hase</eintrag>
</liste>
</entry></row>
<row><entry>Zelle 3</entry><entry></entry></row>
</table>
<liste>
<eintrag>Stein</eintrag>
<eintrag>Haus</eintrag>
</liste>
```

#### Skript:

```
ELEMENT entry
BEG="<td>"
BEG="</td>"
ENDE

ELEMENT row
BEG="<tr>"
END="</tr>"
ENDE

ELEMENT table
BEG="<table>"
END="</table>"
ENDE

ELEMENT liste
BEG="<ul>"
IF INX table
```

```
BEG=' class="tabellenliste" '
ENDIF
END="/ul>"
ENDE

ELEMENT eintrag
BEG="<li>"
END="</li>"
ENDE
```

**Ausgabe:**

```
<table>
<tr><td>Zelle 1</td><td>
<ul class="tabellenliste">
<li>Maus</li>
<li>Hase</li>
</ul></td></tr>
<tr><td>Zelle 3</td><td>&nbsp;</td></tr>
</table>
<ul>
<li>Stein</li>
<li>Haus</li>
</ul>
```

**Skript:**

In diesem Beispiel wird die Liste innerhalb der Tabelle zusätzlich mit dem Attribut `class` und dem Wert `tabellenliste` versehen. Die Liste außerhalb der Tabelle erhält kein Attribut `class`. Die Abfrage `INX` prüft, ob die Liste `liste` ein Unterelement von `table` ist, unabhängig der Verschachtelungstiefe.

**Siehe auch**

AFTER, BEFORE, BEG, EMPTY, END, FIND, FIRST, IN, LAST, SUB, TOP-LEVEL, *Bedingungsausdruck*

## LAST

---

Prüft, ob das aktuelle *Element* das letzte Element innerhalb des übergeordneten *Elements* ist.

### Syntax

#### LAST

### Syntaxbeispiele

```
IF LAST
IF IN eintrag LAST
IF NOTLAST
IF IN eintrag NOTLAST
```

### Typ

#### Bedingung

### Beispiel

#### Daten:

```
<dokument>
<liste>
<eintrag>Elefanten</eintrag>
<eintrag>Tiger</eintrag>
<eintrag>Affen</eintrag>
</liste>
</dokument>
```

#### Skript:

```
ELEMENT dokument
BEG="<html><body>"
END="</body></html>"
ENDE
```

```
ELEMENT liste
BEG="<ul>"
END="</ul>"
ENDE
```

```
ELEMENT eintrag
BEG="<li>"
```

IF **LAST**

```
    END=" (LAST) "
```

```
ENDIF
```

```
IF NOTLAST
```

```
    END=" (NOTLAST) "
```

```
ENDIF
```

```
END="</li>"
```

```
ENDE
```

**Ergebnis:**

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

```
<ul>
```

```
<li>Elefanten (NOTLAST)</li>
```

```
<li>Tiger (NOTLAST)</li>
```

```
<li>Affen (LAST)</li>
```

```
</liste>
```

```
</ul>
```

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

Dieses Beispiel wird mit `LAST` geprüft, ob der jeweilige Listeneintrag eintrag der letzte Eintrag der Liste `liste` ist. In diesem Fall wird der Text "`(LAST)`" angehängt.

**Siehe auch**

AFTER, BEFORE, BEG, EMPTY, END, FIND, FIRST, IN, INX, SUB, TOP-LEVEL, *Bedingungsausdruck*

## MODEL

---

Definiert den Typ des Inhaltsmodells für das betreffende *Element*.

### Syntax

**MODEL**=*Typ*

*Typ*

Gibt den Typ des Inhaltsmodells wie folgt an:

**EMPTY** Das *Element* enthält keine Daten oder Inhalte.

Bitte beachten Sie, dass *EMPTY-Elemente* keinen Endtag besitzen dürfen.

**DATA** Der Inhalt des *Elements* besteht aus PCDATA bzw. RCDATA.

**GROUP** Der Inhalt des *Elements* besteht aus einer Gruppe von *Unterelementen*. Im direkten Inhalt sind keine Daten zugelassen.

**MIXED** Das *Element* kann aus einer beliebiger Kombination von *Unterelementen* und Daten bestehen.

Wichtiger Hinweis zu SGML-DTDs: Beachten Sie hierbei auch die Einschluss- und Ausschlussregeln (inclusions und exclusions).

Im Zweifelsfall geben Sie für leere *Elemente* den `Typ=EMPTY` und für alle anderen *Elemente* den `Typ=MIXED` an.

### Beispiel 1

```
ELEMENT kapitel
```

```
MODEL=GROUP
```

```
ENDE
```

Das Element `kapitel` wird mit dem Modell `GROUP` definiert. Das Element darf also nur Unterelemente, aber selbst kein PCDATA enthalten.

### Beispiel 2

```
ELEMENT bild
```

```
MODEL=EMPTY
```

```
ENDE
```

Das Element `bild` wird mit dem Modell `EMPTY` definiert. Das Element wird hiermit als leeres Element deklariert.

### Siehe auch

`ELEMENT`, *Skript*, *Element-Handler*

## NAMECASE

---

Mit NAMECASE wird die Groß-/Kleinschreibung der Bezeichner (Element- und Attributnamen) gesteuert. Als Voreinstellung wird hierbei für SGML die Groß-/Kleinschreibung ignoriert und für XML als relevant betrachtet.

### Syntax

**NAMECASE** = {**YES**|**NO**}

YES	Ignoriere Groß-/Kleinschreibung. Voreinstellung für SGML-Syntax.
NO	Beachte Groß-/Kleinschreibung. Voreinstellung für XML-Syntax.

### Hinweis

Diese Einstellung entspricht der Einstellung `NAMECASE GENERAL` der SGML-Deklaration:

**NAMECASE GENERAL=YES**

**NAMECASE GENERAL=NO**

Die Einstellung `NAMECASE ENTITY` ist stets `NO`, d.h. für Entitäten wird die Groß-/Kleinschreibung stets beachtet.

Siehe hierfür auch SGML-Deklaration und NAMECASE.

### Beispiel 1

```
SYNTAX=SGML
```

```
NAMECASE=NO
```

Es soll die SGML-Syntax verwendet werden, wobei die Groß-/Kleinschreibung der Bezeichner wie bei XML beachtet bzw. unterschieden werden soll.

### Beispiel 2

```
SYNTAX=XML
```

```
NAMECASE=NO
```

Es soll die XML-Syntax verwendet werden, wobei die Groß-/Kleinschreibung abweichend von der Norm ignoriert werden soll.

### Siehe auch

*Skript, Skript-Einstellungen*

## NOT

---

Präfixoperator. Negation des folgenden Ausdrucks. Der Wahrheitswert des folgenden Befehls wird negiert.

### Syntax

**NOT**AFTER

**NOT**AFTERX

**NOT**BEFORE

**NOT**BEFOREX

**NOT**EMPTY

**NOT**FIND

**NOT**FIRST

**NOT**IN

**NOT**INX

**NOT**LAST

**NOT**SUB

**NOT**TOPLEVEL

### Hinweis

Der Präfix NOT wird direkt dem zu negierenden Befehl vorangestellt. Es dürfen also keine Trennzeichen zwischen NOT und dem Befehl stehen:

- NOT AFTER (falsch!)
- **NOT**AFTER (richtig!).

Bei der Auswertung des Ausdruckes wird nur der verbundene Befehl begiert, nicht aber der darauf folgende Ausdruck:

```
ELEMENT p
IF IN kapitel NOTFIRST IN abschnitt
  ERR="P im Kapitel (nicht das 1.) im Abschnitt."
ENDIF
ENDE
```

bedeutet: Befindet sich das Element p direkt innerhalb des Elementes kapitel, welches nicht das erste Element kapitel ist und welches sich direkt im Element abschnitt befindet.

## Typ

### Präfix einer Bedingung

#### Beispiel 1

```
ELEMENT p
IF NOTEMPTY
  BEG="<p>"
  END="</p>"
ENDIF
ENDE
```

Dieses Beispiel gibt das Element `p` nur dann aus, wenn es auch Daten enthält.

#### Beispiel 2

##### Daten:

```
<p>Das ist <fn><i><b>wichtig</b></i></fn></p>
<p><i>sehr</i> <b>wichtig</b></p>
```

##### Skript:

```
ELEMENT i
IF NOTINX fn
  BEG="<i>"
  END="</i>"
ENDIF
ENDE

ELEMENT b
IF NOTINX fn
  BEG="<b>"
  END="</b>"
ENDIF
ENDE

ELEMENT fn
BEG="<span>"
END="</span>"
ENDE
```

##### Ausgabe:

```
<p>Das ist <span>wichtig</span></p>
```

```
<p><i>sehr</i> <b>wichtig</b></p>
```

Dieses Beispiel gibt die Element `b` und `i` nur dann aus, wenn sie sich nicht in einer Fußnote `fn` befinden und zwar unabhängig der Verschachtelungstiefe.

**Siehe auch**

AFTER, BEFORE, BEG, EMPTY, END, FIND, FIRST, IN, INX, LAST, SUB, TOPLEVEL, *Bedingungsausdruck*



**NOTAFTER**

---

Negation von AFTER.

**Siehe**  
AFTER



## **NOTAFTERX**

---

Negation von AFTERX.

**Siehe**

AFTERX



## **NOTBEFORE**

---

Negation von BEFORE.

**Siehe**

BEFORE



## **NOTBEFOREX**

---

Negation von BEFOREX.

**Siehe**

BEFOREX



**NOTEEMPTY**

---

Negation von EMPTY.

**Siehe**

EMPTY



## **NOTFIND**

---

Negation von FIND.

**Siehe**

FIND



## **NOTFIRST**

---

Negation von FIRST.

**Siehe**

FIRST



**NOTIN**

---

Negation von IN.

**Siehe**

IN



**NOTINX**

---

Negation von INX.

**Siehe**

INX



## **NOTLAST**

---

Negation von LAST.

**Siehe**

LAST



## **NOTSUB**

---

Negation von SUB.

**Siehe**

SUB



## **NOTTOPLEVEL**

---

Negation von TOPLEVEL.

**Siehe**

TOPLEVEL

## **PUBLIC**

---

Definiert den Identifikator (public identifier) des XML-/SGML-Dokumentes.

Dieser Eintrag wird nur dann ausgewertet, wenn das XML-/SGML-Dokument einen DOCTYPE-Eintrag enthält. Im DOCTYPE-Eintrag kann der Public Identifikator definiert werden.

Wenn die Instanzen einen Public Identifikator besitzen, wird dieser mit dem im Skript definierten Public Identifikator verglichen. Eine Konvertierung erfolgt nur dann, wenn beide Identifikatoren übereinstimmen!

### **Syntax**

**PUBLIC** = *Text*

*Text* eine *Zeichenkette*, die einen gültigen XML-/SGML-PUBLIC Identifikator enthält.

### **Beispiel**

```
PUBLIC = "-//Meine Firma//Projekt DTD Version 1//de"
```

Es werden nur Instanzen verarbeitet, deren PUBLIC-Identifikator mit dieser Angabe übereinstimmt. Ist zusätzlich zum PUBLIC-Identifikator ein SYSTEM-Identifikator in den Daten angegeben, so muss zusätzlich der SYSTEM-Identifikator der Daten mit dem SYSTEM-Identifikator des Skriptes übereinstimmen!

### **Siehe auch**

SYSTEM, *Skript*, *Skript-Einstellungen*, *Zeichenkette*

## READERENCODING

---

Definiert die Kodierung der Eingabedaten. Entspricht die Kodierung der Daten nicht dem angegebenen Kodierung, wird die Konvertierung mit Fehlermeldung abgebrochen.

### Syntax

**READERENCODING = { AUTO | ISO-8859-1 | ASCII | US-ASCII | UTF-8 | UTF-16 | UTF-16LE | UTF-16BE | UTF-32 | UTF-32LE | UTF-32BE | BIN | BIN-16 | BIN-16LE | BIN-16BE | BIN-32 | BIN-32LE | BIN-32BE | UCS-2 | UCS-2LE | UCS-2BE | UCS-4 | UCS-4LE | UCS-4BE }**

AUTO                      Automatische Erkennung.

### Syntaxbeispiele

```
READERENCODING=AUTO
```

```
READERENCODING=UTF-8
```

## SUB

---

Prüft, ob das gerade bearbeitete *Element* von einer Unterfunktion einer anderen Elementbearbeitung aufgerufen wird. Diese Unterfunktion wird durch die Funktionen COPY, CUT, FINDONE und FINDALL jeweils bei der Umsetzung nach *Unterelementen* generiert.

### Syntax 1

#### SUB

### Syntax 2

#### SUB(Kontextname)

##### *Kontextname*

Ein *Bezeichner*, der zur Identifizierung einer bestimmten Unterfunktion (COPY oder CUT) verwendet wird. Die Bedingung prüft, ob der angegebene Kontextname mit dem Kontextnamen der aufrufenden Funktion übereinstimmt. Wenn in der aufrufenden Funktion kein Kontextbezeichner angegeben wurde, so wird der Kontextname mit dem *Elementnamen*, in dem sich die aufrufende Funktion befindet, verglichen.

Verwenden Sie diese Funktion, um unterschiedliche Ausgabeformen für COPY, CUT, etc. und der normalen Elementumsetzung zu erzeugen.

### Typ

#### *Bedingung*

### Beispiel

#### Daten:

```
<dokument>
<titel>Vorwort<fn>Bitte beachten!</fn></titel>
<p>Das ist wichtig<fn>Wirklich</fn></p>
</dokument>
```

#### Skript:

```
ELEMENT dokument
BEG="<html><body>"
END="<hr>"+COPY(fn,ALL,FULL)
END="</body></html>"
ENDE

ELEMENT fn
```

```
IF SUB // durch COPY aufgerufen?
```

```
  BEG="<div>* "
```

```
  END="</div>"
```

```
ELSE
```

```
  BEG="*" // ohne COPY
```

```
  DAT=DROP
```

```
ENDIF
```

```
ENDE
```

```
ELEMENT titel
```

```
BEG="<h1>"
```

```
END="</h1>"
```

```
ENDE
```

```
ELEMENT p
```

```
BEG="<p>"
```

```
END="</p>"
```

```
ENDE
```

**Ergebnis:**

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

```
<h1>Vorwort*</h1>
```

```
<p>Das ist wichtig*</p>
```

```
<hr>
```

```
<div>* Bitte beachten!</div>
```

```
<div>* Wirklich</div>
```

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

Dieses Beispiel kopiert alle Fußnoten (Elemente *f<sub>n</sub>*) an das Ende des Dokumentes. Innerhalb des Elementes *f<sub>n</sub>* wird mit **SUB** abgefragt, ob das Element vom **COPY**-Befehl aufgerufen wurde. In diesem Fall wird der Inhalt ausgegeben. Andernfalls wird mit **DAT=DROP** die Ausgabe des Elementinhaltes unterdrückt.

**Siehe auch**

AFTER, BEFORE, BEG, COPY, CUT, EMPTY, END, FIND, FIRST, IN, INX, LAST, TOPLEVEL, *Bedingungsausdruck*

## SYNTAX

---

Schaltet den internen Parser auf die gewünschte Syntax.

### Syntax

**SYNTAX** = {XML|SGML|HTML}

XML	Voreinstellung. XML-Syntax.
SGML	SGML-Syntax gemäß Reference Concrete Syntax.
HTML	HTML5-Syntax. Es wird die pure HTML-Syntax ohne XML-Erweiterung unterstützt.

### Beispiel 1

**SYNTAX**=XML

Es soll die XML-Syntax verwendet werden.

```
<!DOCTYPE dokument SYSTEM "meine.dtd" [  
  <!ENTITY entity1 SYSTEM "bild.jpg" NDATA jpeg>  
>  
<dokument>  
<text>Ein Textbeispiel</text>  
<bild entity="entity1"><!-- Leerelement -->  
</dokument>
```

### Beispiel 2

**SYNTAX**=SGML

Es soll die SGML-Syntax verwendet werden.

```
<!DOCTYPE dokument SYSTEM "meine.dtd">  
<dokument>  
  <abschnitt>  
    <text>Ein Textbeispiel</text>  
    <bild quelle="bild.jpg"/><!-- Leerelement -->  
  </abschnitt>  
</dokument>
```

**Beispiel 3****SYNTAX**=HTML

Es soll die SGML-Syntax verwendet werden.

```
<!DOCTYPE html>
<html>
  <section>
    <p>Ein Textbeispiel</p>
    <!-- Leerelement -->
  </section>
</html>
```

**Siehe auch***Skript, Skript-Einstellungen*

## SYSTEM

---

Definiert den DTD-Dateinamen (system identifier) des XML-/SGML-Dokumentes.

Dieser Eintrag wird nur dann ausgewertet, wenn das XML- oder SGML-Dokument einen DOCTYPE-Eintrag enthält. Im DOCTYPE-Eintrag kann der System Identifikator definiert werden. Der System Identifikator enthält den Dateinamen, unter dem die passende DTD im System zu finden ist.

Wenn die Instanzen einen System Identifikator besitzen, wird dieser mit dem im Skript definierten System Identifikator verglichen. Eine Konvertierung erfolgt nur dann, wenn beide Identifikatoren übereinstimmen!

### Syntax

**SYSTEM** = Zeichenfolge

*Zeichenfolge*

eine *Zeichenkette*, die einen gültigen XML- oder SGML-System Identifikator enthält. Dieser Wert ist ein gültiger Dateiname.

### Beispiel 1

Daten:

```
<!DOCTYPE p SYSTEM "meine.dtd">
<p>
</p>
```

Skript:

```
BASE = "dokument"
```

```
SYSTEM = "meine.dtd"
```

In diesem Beispiel werden nur die Instanzen verarbeitet, deren SYSTEM-Identifikator mit dieser Angabe übereinstimmt.

### Beispiel 2

Daten:

```
<!DOCTYPE p PUBLIC "-//Mein//Projekt DTD Version 1//de"
"meine.dtd">
<p>
</p>
```

Skript:

```
BASE = "dokument"
```

```
SYSTEM = "meine.dtd"
```

```
PUBLIC = "-//Mein//Projekt DTD Version 1//de"
```



In diesem Beispiel werden nur die Instanzen verarbeitet, deren SYSTEM-Identifikator mit dieser Angabe übereinstimmt und wenn der PUBLIC-Identifikator mit der PUBLIC-Identifikator des Skriptes übereinstimmt!

**Siehe auch**

PUBLIC, *Skript*, *Skript-Einstellungen*, *Zeichenkette*

## TOKEN

---

Ausgabe von Token-Attributwerten. Sucht das angegebene Token-Attribut im aktuellen *Element* oder im angegebenen *Element* im erweiterten Kontext und gibt entsprechend der angegebenen Operation den Inhalt aus.

Es wird generell von numerischen Tokens ausgegangen!

Eine Entitäten-Ersetzung erfolgt nicht. Entitäten werden ignoriert bzw. als Leerzeichen interpretiert.

### Syntax 1

**TOKEN**(*[Attributname]*,*Operation*,*Separatortext*)

### Syntax 2

**TOKEN**(*Elementname*[*Attributname*],*Operation*,*Separatortext*)

*Elementname*      Optionaler *Bezeichner* für ein *Element*. Dieses *Element* kann in einem beliebigen Kontext zu dem aktuellen *Element* stehen. Die Kontextzuordnung erfolgt entsprechend dem INX-Kommando.

*Attributname*      *Bezeichner* für den Namen eines *Attributes*. Der *Bezeichner* muss in eckige Klammern gesetzt werden, z.B. [width].

*Operation*          *Zeichenkette*, die durch Hochkommas (" . . ." oder ' . . . ' ) angegeben werden muss.

Dieser Text enthält die Informationen zur Umrechnung der Tokenwerte.

Folgende interne Syntax innerhalb der *Zeichenkette* wird verwendet:

{*number*}

Extrahiert Zahlen aus dem String.

oder

{*Funktion*:}{*Faktor*?}{*Einheit*?}{:*Offset*?}{(*Format*)?}

*Funktion*      Angabe einer Funktion:

**S**      (Summenberechnung aller Tokeneinträge)

Bitte achten Sie auf die Eingabe des Doppelpunktes am Ende der Funktion!

*Faktor*      Ein numerischer Wert, der als Korrekturfaktor für jeden gefundenen Tokeneintrag verwendet wird.

*Einheit*      Angabe einer neuen Maßeinheit, in die jeder gefundene Tokeneintrag

umgerechnet wird. Folgende Maßeinheiten werden als Umrechnungseinheit unterstützt:

- pt** Punkt
- mm** Millimeter
- cm** Zentimeter
- in** Inch, Zoll

Die Einheit selbst wird im Ergebnis **nicht** ausgegeben!

Die Berechnung relativer Werte erfolgt mit:

- %** Umrechnung in Prozent bzw.
- %%** wie % aber mit zusätzliche Ausgabe des Prozentzeichens.

*Offset*

Ein numerischer Wert, der als Korrekturoffset für jeden gefundenen Tokeneintrag verwendet wird. Folgende Werte sind möglich:

- +** numerischer Wert für ein positives Offset
- numerischer Wert für ein negatives Offset.

Bitte achten Sie auf die Eingabe des Doppelpunktes am Beginn des Offsets!

*Format*

Definition des Ausgabezahlenformates. Bitte stets die Klammern setzen, z.B. (**# . ##**). Folgende Platzhalterzeichen können verwendet werden:

- #** Platzhalter für eine Dezimalzahl
- .** Dezimalpunkt

Beispiele:

Beispiele	<b>0</b>	<b>12</b>	<b>5467,872</b>	<b>0,6785</b>
<b>#</b>	0	12	5468	1
<b>##</b>	0,00	12,00	5467,87	0,68
<b>###</b>	000	012	5468	001
(ohne)	0	12	5467,872	0,6785

Wenn kein Format definiert wurde, gelten folgende Voreinstellungen:

<b>pt</b>	#.#
<b>mm</b>	#.##
<b>cm</b>	#.###
<b>in</b>	#.####
<b>%</b>	#.#
<b>%%</b>	#.#
-	(automatisch)

Wenn keine Operation angegeben wird, erfolgt auch keine Umrechnung und Formatierung!

Wenn kein Format angegeben wird, werden die Dezimalstellen automatisch erstellt (Ganzzahlen ohne Dezimalpunkt, ansonsten bis zu 8 Dezimalstellen).

*Separatortext*

*Zeichenkette, die eine Text enthält, der zwischen die gefundenen Token-Einträge eingefügt wird.*

### Hinweis

Wenn Sie keine numerischen Daten bearbeiten wollen bzw. keine Operation angeben haben, verwenden Sie stattdessen IMPLIED oder VALUE.

### Typ

*Anweisung*

### Beispiel

Daten:

```
<table colwidths="100 300 100">
</table>
```

Skript:

```
ELEMENT table
BEG='<table width="100%">'
BEG='<col width="' +TOKEN([colwidths], "%", "'>
<col width="' +TOKEN([colwidths], "%", "'>'>'
END='</table>'
```

Ausgabe:

```
<table width="100%">
<col width="20%">
<col width="60%">
<col width="20%">
</table>
```



In diesem Beispiel wird das Tabellenattribut `colwidths` (NMTOKEN-Version) ausgelesen und die einzelnen Tokenwerte prozentual umgerechnet. Die einzelnen Werte werden als Spaltendefinitionen ausgegeben.

**Siehe auch**

BEG, CHR, COPY, COUNT, CUT, DEL, END, FILE, IMPLIED, VALUE, *Element-Konvertierungsregel*, *Zeichenkette*

## TOPLEVEL

---

Prüft, ob das aktuelle *Element* das Toplevel-*Element* der Instanz ist. Das Toplevel-*Element* ist hierbei das oberste *Element* im Dokument.

### Syntax

#### TOPLEVEL

### Typ

#### Bedingung

### Beispiel

#### Daten:

```
<?xml version="1.0"?>
<kapitel>
<titel>Bedienungsanleitung</titel>
<p>Das ist sehr <b>wichtig</b></p>
</kapitel>
```

#### Skript:

```
ELEMENT kapitel
IF TOPLEVEL
  BEG='<!DOCTYPE html>'
  BEG='<html>'
  END='</html>'
ENDIF
ENDE
```

#### Ergebnis:

```
<!DOCTYPE html>
<html>
...
</html>
```

Wenn das *Element* `kapitel` das oberste *Element* ist, wird der HTML-Doctype und das HTML-*Element* ausgegeben.

### Siehe auch

AFTER, BEFORE, BEG, EMPTY, END, FIND, FIRST, IN, INX, LAST, SUB, *Bedingungsausdruck*

## VALUE

---

Verfügbar als Anweisung (siehe VALUE) und als Bedingung (siehe VALUE).

### VALUE-Anweisung

---

Ausgabe eines *Attributwertes*. Sucht das angegebene Attribut im aktuellen *Element* oder im angegebenen *Element* im erweiterten Kontext und gibt den Inhalt aus.

#### Syntax 1

**VALUE**([*Attributname*])

#### Syntax 2

**VALUE**([*Attributname*],[*Operation*|*Ersetzungstyp*])

#### Syntax 3

**VALUE**(*Elementname*[*Attributname*])

#### Syntax 4

**VALUE**(*Elementname*[*Attributname*],[*Operation*|*Ersetzungstyp*])

*Elementname*      Optionaler *Bezeichner* für ein *Element*. Dieses *Element* kann in einem beliebigen Kontext zu dem aktuellen *Element* stehen. Die Kontextzuordnung erfolgt entsprechend dem INX-Kommando.

*Attributname*      *Bezeichner* für den Namen eines *Attributes*.

*Operation*      Optionale *Zeichenkette* für eine Umrechnungsoperation und/oder Formatierung des Attributwertes. Diese Zeichenkette muss in Anführungszeichen ("..." oder '...') angegeben werden.

Folgende interne Syntax innerhalb der *Zeichenkette* wird verwendet:

$\{Faktor\}?\{Einheit\}?\{Offset\}?\{(Format)\}?$

*Faktor*      Ein numerischer Wert, der als Korrekturfaktor für den Attributwert verwendet wird.

*Einheit*      Angabe einer neuen Maßeinheit, in die der Attributwert umgerechnet wird. Folgende Maßeinheiten werden als Umrechnungseinheit unterstützt:

**pt**      Punkt

**mm**      Millimeter

**cm** Zentimeter

**in** Inch, Zoll

Die Einheit selbst wird im Ergebnis **nicht** ausgegeben!

Achtung: Eine Umrechnung erfolgt nur dann, wenn der Attributwert ebenfalls mit einer Maßeinheit versehen ist!

*Offset*

Ein numerischer Wert, der als Korrekturoffset für den Attributwert verwendet wird. Folgende Werte sind möglich:

**+** numerischer Wert für ein positives Offset

**-** numerischer Wert für ein negatives Offset.

Bitte achten Sie auf die Eingabe des Doppelpunktes am Beginn des Offsets!

*Format*

Definition des Ausgabezahlenformates. Bitte stets die Klammern setzen, z.B. (# . ##). Folgende Platzhalterzeichen können verwendet werden:

**#** Platzhalter für eine Dezimalzahl

**.** Dezimalpunkt

Beispiele:

Beispiele	<b>0</b>	<b>12</b>	<b>5467,872</b>	<b>0,6785</b>
<b>#</b>	0	12	5468	1
<b>##</b>	0,00	12,00	5467,87	0,68
<b>###</b>	000	012	5468	001
<i>(ohne)</i>	0	12	5467,872	0,6785

Wenn kein Format definiert wurde, gelten folgende Voreinstellungen, wenn eines der Zielformate angegeben wurde:

**pt** # . #

**mm** # . ##

**cm** # . ###

**in** # . ####

**%** # . #

%% #.#

- (automatisch)

Wenn keine Operation angegeben wird, erfolgt auch keine Umrechnung und Formatierung!

Wenn kein Format angegeben wird, werden die Dezimalstellen automatisch erstellt (Ganzzahlen ohne Dezimalpunkt, ansonsten bis zu 8 Dezimalstellen).

### Ersetzungstyp

Gibt den Ersetzungstyp für die Daten an. Es stehen folgende Varianten zur Auswahl:

**DATA** Die enthaltenen Entitäten werden nach den definierten Ersetzungsregeln (siehe ENTITY) umgesetzt.

**ALT** Enthaltene Entitäten werden alternativ (siehe ALTERNATE) umgesetzt.

**ASC** Enthaltene Entitäten werden nach ASCII (siehe ASCII) umgesetzt.

Wenn kein Wert angegeben wird, wird der Ersetzungstyp **DATA** als Standard verwendet.

Wenn statt dem Ersetzungstyp eine *Operation* angegeben wird, werden die Daten ohne Entitätenersetzung konvertiert. Die *Operation* geht in der Regel von numerischen Daten aus!

### Syntaxbeispiele

```
BEG=VALUE ([typ])
```

```
BEG=VALUE (abschnitt [typ])
```

```
BEG=VALUE ([width], "#.##")
```

```
BEG=VALUE ([width], "1.5: (#.##)")
```

```
BEG=VALUE ([width], "1.5pt:+100 (#.##)")
```

```
BEG=VALUE (entry [width], "#.##")
```

```
BEG=VALUE (entry [width], "1.5: (#.##)")
```

```
BEG=VALUE (entry [width], "1.5pt:+100 (#.##)")
```

```
BEG=VALUE (abschnitt [kategorie], ASC)
```

```
BEG=VALUE (abschnitt [kategorie], ALT)
```

### Typ

#### Anweisung

#### Beispiel 1

Daten:

```
<kapitel nr="5">
</kapitel>
```

**Skript:**

```
ELEMENT kapitel
  ATTLIST [nr=""]
  BEG='<DIV><A NAME="'+VALUE ([nr]) + '"/>'
  END="</DIV>"
ENDE
```

**Ergebnis:**

```
<DIV><A NAME="5"/></DIV>
```

Mit der Attributabfrage (VALUE) wird der Wert des Attributes `nr` des *Elementes* `kapitel` abgefragt und als HTML-Ankerelement `a` mit dem Attribut `name` ausgegeben.

**Beispiel 2****Daten:**

```
<kapitel nr="5&sol;4">
  <titel>Vorwort</titel>
</kapitel>
```

**Skript:**

```
ELEMENT titel
  BEG="<h1>"+VALUE (kapitel [nr]) + " "
  END="</h1>"
ENDE
```

**Ergebnis:**

```
<H1>5&sol;4 Vorwort</H1>
```

Mit der Attributabfrage (VALUE) wird der Wert des Attributes `nr` des *Elementes* `kapitel` abgefragt und als Text der HTML-Überschrift `h1` ausgegeben.

**Hinweis**

Rekursionsschutz! Bei Verwendung von VALUE innerhalb einer *Entitäten-Ausgabeangabe* werden nur Textbestandteile (CDATA) des betreffenden *Attributwerts* zurückgegeben und wiederum enthaltene Entitäten ausgelassen!

**Siehe auch**

BEG, CHR, COPY, COUNT, CUT, DEL, END, FILE, IMPLIED, TOKEN, *Element-Konvertierungsregel*, *Zeichenkette*



## VALUE-Bedingung

---

Abfrage eines Attributwertes.

Diese Funktion entspricht der *Attributbedingung*.

### Syntax 1

**[Attributname Operator Wert]**

Vereinfachte Syntax ohne das Schlüsselwort VALUE, siehe auch *Attributbedingung*.

### Syntax 2

**VALUE([Attributname Operator Wert])**

### Syntax 3

**VALUE(Elementname[Attributname Operator Wert])**

*Elementname*            Optionaler *Bezeichner* für ein *Element*. Dieses *Element* kann in einem beliebigen Kontext zu dem aktuellen *Element* stehen. Die Attributabfrage wird auf dieses Element angewendet.

*Operator*            Vergleichsoperator:

<	kleiner
>	größer
<=	kleiner gleich
>=	größer gleich
<>	ungleich (unabhängig der Groß- und Kleinschreibung)
=	gleich (unabhängig der Groß- und Kleinschreibung)
==	identisch (mit Beachtung der Groß- und Kleinschreibung)
!=	nicht identisch (mit Beachtung der Groß und Kleinschreibung)

*Wert*                 *Zeichenkette*, für den *Attributwert*.

### Syntaxbeispiele

```
IF VALUE ([typ<>""])
IF VALUE (abschnitt[typ=="vorwort"])
IF VALUE (abschnitt[typ!="vorwort"])
IF VALUE (abschnitt[typ<>"vorwort"])
```

```
IF [typ<>""]  
IF [typ=="vorwort"]
```

## Typ

### Bedingung

### Beispiel

#### Daten:

```
<entry align="right">  
  <p>100<p>  
</entry>
```

#### Skript:

```
ELEMENT entry  
IF VALUE([align="right"])  
  BEG='<TD CLASS="RIGHT">  
  END='</TD>'  
ENDE
```

Diese Bedingung prüft, ob die Eigenschaft `align` den Wert `right` besitzt.

### Siehe auch

**IMPLIED**, *Attributbedingung*, *Attributname*, *Attributwert*

## WARNINGS

---

Erlaubt die Steuerung der Datenprüfung. Per Voreinstellung führen alle Fehler (Fehlermeldungen) zum Abbruch der Konvertierung. Mit dem Schalter (WARNINGS=SMART) kann aber die Fortsetzung der Konvertierung für einige Fehler angewiesen werden. In diesem Modus werden die Eingabedaten auch nicht auf Kodierungsfehler geprüft.

Folgende Fehler führen nicht zum Abbruch im Modus SMART:

- nichtdeklarierte Elemente
- nichtdeklarierte Entitäten
- nichtdeklarierte Zeichenverweise.

### Syntax

**WARNINGS = {SMART|ALL}**

ALL	Voreinstellung. Alle Fehler führen zum Abbruch der Operation.
SMART	SGML-Syntax gemäß Reference Concrete Syntax.

### Beispiel

**SYNTAX=SMART**

Es wird keine Codeprüfung der Eingabedaten vorgenommen. Fehlermeldungen für nichtdeklarierte Elemente, Entitäten und Zeichenreferenzen führen nicht zum Abbruch der Konvertierung.

### Siehe auch

*Skript, Skript-Einstellungen*

## WRITERENCODING

---

Definiert die Kodierung der Ausgabedaten.

### Syntax

**WRITERENCODING** = { **AUTO** | **ASCII** | **BIN** | **UTF-8** | **UTF-16** | **UTF-16LE** | **UTF-16BE** | **UTF-32** | **UTF-32LE** }

AUTO	Automatische Erkennung. Die Kodierung der zu lesenden Daten wird für die Ausgabe verwendet.
ASCII	7 Bit ASCII Alle ASCII-Zeichen im Bereich 0...127 werden unverändert ausgegeben. Zeichen ab 128 (x80 ... x10FFFF) werden als Zeichenverweise <code>&amp;#x80;</code> kodiert.
BIN	8 Bit binär Alle ASCII-Zeichen im Bereich 0...255 werden unverändert ausgegeben. Zeichen ab 256 (x100 ... x10FFFF) werden als Zeichenverweise <code>&amp;#x100;</code> kodiert.
UTF-8, UTF-16, ...	normgerechte UTF-Kodierungen

### Hinweis

Verwenden Sie `AUTO` nicht, wenn Sie Ausgangsdaten mit unterschiedlichen Kodierungen verwenden bzw. die nicht sicherstellen können, dass die Ausgangsdaten exakt die gewünschte Kodierung verwendet!

### Syntaxbeispiele

```
WRITERENCODING=BIN
```

```
WRITERENCODING=UTF-8
```

**X**

---

Suffixoperator. Kontexterweiterung des voranstehenden Befehls. Es können im Kontext davor, danach oder darüber noch beliebige Elemente dazwischenstehen.

**Syntax**

AFTERX

BEFOREX

INX

NOTAFTERX

NOTBEFOREX

NOTINX

**Hinweis**

Der Suffix X wird direkt an den zu negierenden Befehl angefügt. Es dürfen also keine Trennzeichen zwischen X und folgendem Befehl stehen.

## Fehlermeldungen

Der Converter gibt den Erfolg der Programmausführung als Errorlevel zurück:

- |    |  |
|----|--|
| 0  | Erfolg, keine Fehler aufgetreten   |
| 1  | Parameter konnten nicht gelesen werden bzw. unbekannter Parameter gefunden |
| 2  | Sonstige Fehler bei der Parametereauswertung                               |
| 3  | Kein Parameter angegeben   |
| 4  | Hilfe wurde angezeigt, es wurde aber keine Konvertierung durchgeführt      |
| 5  | Interpreter konnte nicht initialisiert werden                              |
| 6  | Kein Skript angegeben  |
| 7  | Eingabemuster ungültig   |
| 8  | Eingabedateiname ungültig  |
| 9  | .  |
| 10 | Ausgabedateiname ungültig  |
| 11 | Ausgabedateinamenserweiterung ungültig                                     |
| 12 | Datenfehler  |
| 13 | Sonstige Fehler  |

Weitere detaillierte Fehlermeldungen werden über die Standardausgabe (Konsole) angezeigt.



## XML/SGML-Glossar

Übersicht über die wichtigsten XML-/SGML-Bestandteile in alphabetischer Reihenfolge.

### **Wichtiger Hinweis**

Die Beschreibung ist jeweils nur auszugsweise!

**ANY**

Schlüsselwort im Inhaltsmodell einer Elementdeklaration (siehe ELEMENT). Mit ANY wird im Inhaltsmodell jede beliebige Kombination von Elementen und Zeichendaten deklariert. Einzige Bedingung für korrekten Inhalt ist, dass die verwendeten Elemente innerhalb der DTD deklariert sein müssen.

**Syntax XML:**

<!ELEMENT *name* **ANY** >

*name*                      Bezeichner für ein Element

**Syntax SGML:**

<!ELEMENT *name* **ANY** >

<!ELEMENT (*gruppe*) - o **ANY** >

*name*                      Bezeichner für ein Element

*gruppe*                    Elementgruppe. Ein oder mehrere Elementbezeichnungen, die mit Operatoren verbunden sind. Gültige Operatoren sind:

&                      Und

and                    Und

|                        Oder

or                      Oder

,                        Sequenz

Hinweis: Die Funktion der Operatoren wird in der Elementdeklaration ignoriert. Es werden nur die verwendeten Bezeichnungen der Elemente ausgewertet.

## ATTLIST

---

Deklaration bzw. Vereinbarung von Attributen zu einem Element.

### Syntax XML

**<!ATTLIST** *bezeichner* {*attributname typ vorgabewert*}\* **>**

<i>bezeichner</i>	Ein Name des zugehörigen Elementes: <i>Elementname</i>
<i>attributname</i>	Name eines Attributes:
<i>typ</i>	Typ des Attributes: CDATA ENTITY ENTITIES ID IDREF IDREFS NMTOKEN NMTOKENS NOTATION oder eine Gruppe von Auswahlwerten.
<i>vorgabewert</i>	Vorgabewert: FIXED IMPLIED REQUIRED oder ein Vorgabewert (Zeichenkette).

### Syntax SGML

**<!ATTLIST** *bezeichner* {*attributname typ vorgabewert*}\* **>**

<i>bezeichner</i>	Ein Name oder eine Namensgruppe eines zugehörigen Elementes: <i>Elementname</i> ( <i>Elementname</i>   <i>Elementname</i>  ...)
<i>attributname</i>	Name eines Attributes:
<i>typ</i>	Typ des Attributes: CDATA ENTITY



ENTITIES

ID

IDREF

IDREFS

NAME

NAMES

NMTOKEN

NMTOKENS

NOTATION

NUMBER

NUMBERS

NUTOKEN

NUTOKENS

oder

eine Gruppe von Auswahlwerten.

*vorgabewert*

Vorgabewert:

CONREF

CURRENT

FIXED

IMPLIED

REQUIRED

oder

ein Vorgabewert (Zeichenkette).

**Hinweis (XML Converter):**

Die DTD und deren Bestandteile werden nicht ausgewertet.

## **Attribut**

---

Eigenschaft eines *Elementes*.

Attribute werden innerhalb von *Elementen* angegeben.

### **Syntax**

*Attributname*=*Attributwert*

Attribute sind nur innerhalb von Elementen erlaubt:

**<Elementname{ Attributname=Attributwert}\*{}}\*>**

*Elementname*                    *Bezeichner* für den Namen eines *Elementes*

*Attributname*                   *Bezeichner* für den Namen eines *Attributes*.

*Attributwert*                    *Zeichenkette*.

### **Hinweis**

Es dürfen als Attributwerte nur *Zeichenketten* verwendet werden!

### **Typ**

*Dokument*

**Attributname**

---

Ein Attributname ist ein gültiger *Bezeichner* für ein *Attribut*.

**Typ**

*Skript, Dokument*

**Siehe auch**

*Attributwert, Element*

**Attributwert**

---

Ein gültiger Attributwert ist eine Zeichenfolge, die den Wert eines Attributes darstellt. Dieser Wert wird stets als *Zeichenkette* ausgelesen.

Folgende Zeichen sind generell zu vermeiden:

&lt;

&gt;

Ein Attributwert darf Entitäten enthalten.

**Hinweis (XML Converter):**

Die zu verwendenden Zeichen für einen Attributwert sind in der SGML-Deklaration festgelegt. Es wird die Reference Concrete Syntax von SGML vorausgesetzt, deren Konventionen fest integriert sind.

**Typ**

*Skript, Dokument*

## Ausführungsanweisung

---

### Syntax XML

#### <?Bezeichner Zeichendaten?>

<?	Start der Ausführungsanweisung
<i>Bezeichner</i>	XML- <i>Bezeichner</i>
<i>Zeichendaten</i>	beliebige Zeichen(siehe CDATA), ausgenommen dem ">"-Kennzeichen für das Ende der Ausführungsanweisung. Leerzeichen am Ende der Ausführungsanweisung sind nicht erlaubt!
?>	Ende der Ausführungsanweisung

### Syntax SGML

#### <?Zeichendaten>

<?	Start der Ausführungsanweisung
<i>Zeichendaten</i>	beliebige Zeichen(siehe CDATA), ausgenommen dem ">"-Kennzeichen für das Ende der Ausführungsanweisung. Leerzeichen am Beginn und am Ende der Ausführungsanweisung sind nicht erlaubt!
>	Ende der Ausführungsanweisung

#### Hinweis (XML Converter):

Die maximale Größe der Zeichendaten einer Ausführungsanweisung ist auf 16384 (16 KByte) eingeschränkt.

#### Hinweis (XML Converter):

Ausführungsanweisungen werden nicht ausgewertet!

## Ausschluss

---

### Exclusion

Ausschlüsse sind zusätzlich zu einem Element definierte Elemente, die **nicht** im Elementinhalt an beliebiger Stelle und auch **nicht** in alle enthaltenen Unterelementen vorkommen dürfen.

### Hinweis zum Auflösen von Einschlüssen/Ausschlüssen

Folgende Prioritätsregeln sind bei der Bewertung der Gültigkeit von Ein- und Ausschlüssen unbedingt zu beachten.

- Ausschluss, wenn das zu untersuchende Element im Kontext als Ausschluss definiert wurde, darf es an der aktuellen Stelle nicht positioniert werden.
- Einschluss, wenn das zu untersuchende Element im Kontext als Einschluss definiert wurde, ist es an der aktuellen Stelle erlaubt.
- Element in der Modellgruppe, wenn das zu untersuchende Element weder als Ein- noch als Ausschluss im Kontext definiert wurde, wird die Gültigkeit ausschließlich durch die aktuelle Modellgruppe definiert (Definition in der DTD).

### Hinweis

Gilt nur für SGML.

### Hinweis (XML Converter):

Die DTD und deren Bestandteile werden nicht ausgewertet.

### Siehe auch

Einschluss

## Bereiche, bedingte

---

Bedingte Bereiche oder bedingte Abschnitte.

In XML Definitionsbereiche, die für den Parser ein-/ausgeschlossen behandelt werden. Beachte: Bedingte Bereiche sind nicht in Elementinhalten zulässig!

### Hinweis

Nur in XML verfügbar. In SGML wird diese Funktionalität mit markierten Bereichen realisiert (siehe Bereiche, markierte).

### Syntax

**<![INCLUDE[*Zeichendaten*]]>**

**<![ INCLUDE [*Zeichendaten*]]>**

*Zeichendaten* beliebige Textzeichen, ausgenommen der  
Endekennung " ] ]>".

Bereiche, die vom Parser eingeschlossen werden.

**<![IGNORE[*Zeichendaten*]]>**

**<![ IGNORE [*Zeichendaten*]]>**

*Zeichendaten* beliebige Textzeichen, ausgenommen der  
Endekennung " ] ]>".

Bereiche, die vom Parser ausgeschlossen (ignoriert) werden.

### Hinweis

Innerhalb der Parameterseparatoren vor und nach dem Schlüsselwort INCLUDE und IGNORE sind Parameterentitäten erlaubt.

## Bereiche, markierte

---

Markierte Bereiche oder markierte Abschnitte.

In XML Datenbereiche, die ungeparst ausgegeben werden (CDATA). In SGML Datenbereiche, die entweder ungeparst ausgegeben werden (CDATA), deren Entitäten ausschließlich geparst werden (RCDATA), die komplett ausgeschlossen werden (IGNORE) oder die normal geparst werden (INCLUDE und TEMP).

### Syntax XML

**<![CDATA[Zeichendaten]]>**

*Zeichendaten* beliebige Textzeichen, ausgenommen der Endekennung " ] ] > ". Entitäten und Elemente werden hier als normaler Text interpretiert.

### Hinweis

Aus der SGML-Syntax wurden die Typen INCLUDE und IGNORE in einer vergleichbaren Funktionalität übernommen. Allerdings wurden diese in bedingte Bereiche (siehe Bereiche, bedingte) umbenannt. Diese bedingten Bereiche sind im Gegensatz zu SGML aber nicht innerhalb des Elementinhaltes zulässig!

### Syntax SGML

**<![CDATA[Zeichendaten]]>**

**<[ CDATA [Zeichendaten]]>**

*Zeichendaten* beliebige Textzeichen, ausgenommen der Endekennung " ] ] > ". Entitäten und Elemente werden hier als normaler Text interpretiert.

**<![RCDATA[Zeichendaten]]>**

**<[ RCDATA [Zeichendaten]]>**

*Zeichendaten* beliebige Textzeichen, ausgenommen der Endekennung " ] ] > ". Elemente werden hier als normaler Text interpretiert. Entitäten werden vom Parser aber erkannt.

**<![INCLUDE[Zeichendaten]]>**

**<[ INCLUDE [Zeichendaten]]>**

*Zeichendaten* beliebige Textzeichen, ausgenommen der Endekennung " ] ] > ". Entitäten und Elemente werden vom Parser aber erkannt.

**<![IGNORE[Zeichendaten]]>**

**<![ IGNORE [Zeichendaten]]>**

*Zeichendaten* beliebige Textzeichen, ausgenommen der Endekennung " ] ] >". Die Zeichendaten werden vom Parser komplett ignoriert.

**<![TEMP[Zeichendaten]]>**

**<![ TEMP [Zeichendaten]]>**

*Zeichendaten* beliebige Textzeichen, ausgenommen der Endekennung " ] ] >". Entitäten und Elemente werden vom Parser aber erkannt.

### Hinweis SGML

Innerhalb der Parameterseparatoren vor und nach den Schlüsselworten CDATA, RCDATA, INCLUDE, IGNORE und TEMP sind Parameterentitäten erlaubt.

## **Bezeichner**

---

Ein Bezeichner ist eine Zeichenfolge, die den Namen eines Elementes, einer Entität, eines Attributes oder eines Auswahlwertes eines Attributwertes darstellt.

### **XML:**

Das erste Zeichen eines Bezeichners wird wie folgt definiert:

1. Zeichen:            ABCDEFGHIJKLMN**OP**QRSTUVWXYZ  
                          abcdefghijklmnopqrstuvwxyz  
                          :\_  
                          xC0-xD6  
                          xD8-xF6  
                          xF8-xFF

Alle folgenden Zeichen können aus dem erweiterten Zeichenbereich entnommen werden:

folgende Zeichen:    ABCDEFGHIJKLMN**OP**QRSTUVWXYZ  
                          abcdefghijklmnopqrstuvwxyz  
                          0123456789  
                          : . - \_  
                          xC0-xD6  
                          xD8-xF6  
                          xF8-xFF

### **Hinweis (XML Converter):**

Die maximale Länge einer Zeichenkette ist in XML nicht eingeschränkt. Abweichend vom Standard ist hier die maximale Länge eines Bezeichners auf **255 Zeichen** beschränkt worden. Bitte beachten Sie diese Einschränkung!

### **SGML:**

Die zu verwendenden Zeichen für einen Bezeichner sind in der SGML-Deklaration festgelegt. Es wird die Reference Concrete Syntax von SGML voraus, deren Konventionen fest integriert sind. Das erste Zeichen eines Bezeichners wird wie folgt definiert:

1. Zeichen:            ABCDEFGHIJKLMN**OP**QRSTUVWXYZ  
                          abcdefghijklmnopqrstuvwxyz

Alle folgenden Zeichen können aus dem erweiterten Zeichenbereich entnommen werden:

folgende Zeichen:    ABCDEFGHIJKLMN**OP**QRSTUVWXYZ  
                          abcdefghijklmnopqrstuvwxyz  
                          0123456789



Die Länge des Bezeichners ist ebenfalls in der Deklaration festgelegt.

**Hinweis (XML Converter):**

Die Länge des Bezeichners ist auf maximal **255 Zeichen** begrenzt.



---

**BOM**

---

(Byte-Order-Mark)

Mit dem BOM – Byte-Reihenfolge-Markierung – wird in XML-Daten das Auswerteverfahren der Kodierung gesteuert. Das BOM steht am Beginn des Dokumentes und kann folgende Werte haben:

EF BB BF	UTF-8
FE FF	UTF-16 BE
00 00 FE FF	UTF-32 BE
FF FE	UTF-16 LE
FF FE 00 00	UTF-32 LE

Als Standardverfahren wird UTF-8 angenommen.

Wenn das XML-Dokument eine XML-Deklaration enthält, muss die Kodierung des BOM mit der Kodierung der XML-Deklaration übereinstimmen.

## CDATA

---

Zeichendaten (**character data**)

### Syntax 1 - XML

In Attributwerten.

Datenbereich, der Textinformationen und *Entitäten* enthalten darf. Es dürfen in diesem Bereich keine Elemente ("**<**" ...) eingetragen werden.

### Beispiel:

```
<?xml version="1.0"?>
<!DOCTYPE dokument SYSTEM "referenz.dtd">
<dokument>
  <absatz>Ein <b>Text</b> (&#223;).</absatz>
  <absatz synonym="Ein Text (&#223;)."></absatz>
</dokument>
```

### Ausgabe

```
Ein Text (β).
Ein Text (β).
```

### Syntax 2 - SGML

Im Elementinhalt und in Attributwerten.

Datenbereich, der Textinformationen und *Entitäten* enthalten darf. Elemente in diesen Bereich werden als Text interpretiert.

### Beispiel

```
<!DOCTYPE dokument SYSTEM "referenz.dtd">
<dokument>
  <absatz>Ein normaler <b>Text</b> (&#223;).</absatz>
  <absatz synonym="Ein <b>Text</b> (&#223;)."></absatz>
</dokument>
```

### Ausgabe

```
Ein normaler Text (β).
Ein <b>Text</b> (β).
```

### Hinweis (XML Converter):

Elemente sind in diesem Bereich vergleichbar dem XML nicht erlaubt!

### Syntax 3

Innerhalb von markierten Bereichen.

**<![CDATA[*Zeichendaten*]]>**

Datenbereiche, die reine Textinformationen enthalten.

Es erfolgt kein Parsen. Es können beliebige Zeichen verwendet werden, ausgenommen der Endekennung für markierte Bereiche "] ]>". Achtung, Elemente und Entitäten werden in diesem Bereich als Text interpretiert und nicht als *Elemente* und *Entitäten*.

**Beispiel:**

```
<?xml version="1.0"?>
<!DOCTYPE dokument SYSTEM "referenz.dtd">
<dokument>
  <absatz><![CDATA[ <b>Text</b> (&#223;) ]]></absatz>
</dokument>
```

**Ausgabe:**

```
<b>Text</b> (&#223;)
```

**Hinweis:**

Auf die unterschiedliche Interpretation von CDATA ist zu achten! An dieser Stelle ist die SGML- und auch XML-Norm leider uneindeutig. Innerhalb von markierten CDATA-Bereichen werden Elemente und Entitäten als "normaler Text" interpretiert. Attributwerte, die als CDATA deklariert sind, dürfen aber Entitäten enthalten.

## DOCTYPE

---

Vereinbarung des Dokumenttyps. Mit dem Dokumenttyp wird auch das oberste Element im Dokument deklariert. Ebenfalls wird die DTD zum Dokument deklariert, entweder als direkte Angabe einer Datei (SYSTEM) oder als öffentlicher Bezeichner (PUBLIC). Erweiterungen zur DTD werden optional in der erweiterten Deklaration angegeben.

### Syntax 1

```
<!DOCTYPE Elementname SYSTEM System-ID {Erweiterung}?>
```

### Syntax 2 (SGML)

```
<!DOCTYPE Elementname PUBLIC Public-ID {System-ID}? {Erweiterung}?>
```

### Syntax 3 (XML)

```
<!DOCTYPE Elementname PUBLIC Public-ID {System-ID} {Erweiterung}?>
```

### Syntax 4

```
<!DOCTYPE Elementname Erweiterung>
```

<i>Elementname</i>	<i>Bezeichner</i> des obersten <i>Elementes</i> im Dokument (Basiselement).
<i>System-ID</i>	<i>Zeichenkette</i> , die den Namen einer Datei im Dateisystem angibt.
<i>Public-ID</i>	<i>Zeichenkette</i> , die den öffentlichen Bezeichner angibt.
<i>Erweiterung</i>	erweiterte Vereinbarung (Deklaration), die mit [ ... ] gekennzeichnet wird. [           Beginn der erweiterten Vereinbarung ]           Ende der erweiterten Vereinbarung

### Beispiel 1

```
<!DOCTYPE referenz PUBLIC "-//Micha//DTD Referenz Test V1.00//DE">
```

Als oberstes Element im Dokument wird das Element *referenz* deklariert, dessen DTD unter dem öffentlicher Bezeichner "-//Micha//DTD Referenz Test V1.00//DE" dem System bekannt ist.

### Beispiel 2

```
<!DOCTYPE referenz SYSTEM "ref.dtd">
```

Als oberstes Element im Dokument wird das Element `referenz` deklariert, dessen DTD die Datei `ref.dtd` ist.

### Beispiel 3

```
<!DOCTYPE referenz PUBLIC "-//Micha//DTD Referenz Test V1.00//DE" "ref.dtd">
```

Als oberstes Element im Dokument wird das Element `referenz` deklariert, dessen DTD unter dem öffentlicher Bezeichner `"-//Micha//DTD Referenz Test V1.00//DE"` dem System bekannt ist und dessen DTD in der Datei `ref.dtd` zu finden ist.

### Beispiel 4

```
<!DOCTYPE dokument PUBLIC "-//Seume//DTD Handbuch V1.00//DE" [  
<!ENTITY entity.001 SYSTEM "grafik.tif" NDATA bmp>  
>
```

Als oberstes Element im Dokument wird das Element `dokument` deklariert, dessen DTD unter dem öffentlicher Bezeichner `"-//Seume//DTD Handbuch V1.00//DE"` dem System bekannt ist. Zusätzlich wird in der erweiterten Deklaration die externe Grafikdatei `"grafik.tif"` als Entität `"entity.001"` deklariert.

## **Dokument**

---

Das zu verarbeitende XML-/SGML-Dokument.

Nach Norm muss ein Dokument als Gesamtheit aller Definitionen und Daten (Instanz der Definition) betrachtet werden. In der Praxis werden Definitionen und Daten (Instanzen) in separaten Dateien getrennt gespeichert und verwaltet.

Ein Dokument wird in der Regel auch als **Instanz** der DTD bezeichnet.

### **XML:**

Dokument (`xml`)

- BOM?
- Prolog
- Instanz

Extern (`dtd`):

- DTD

Ein XML-Dokument kann von einem BOM (Byte-Reihenfolge-Markierung) eingeleitet werden, welches die Kodierung (UTF-8) oder die Byte-Reihenfolge für UTF-16 oder UTF-32 angibt. Das BOM ist in der Regel in Texteditoren nicht sichtbar! Texteditoren werten das BOM selbst für die korrekte Kodierung aus.

### **SGML-Dokument**

Dokument (`sgm`)

- Prolog
- Instanz

Extern:

- SGML-Deklaration (`dtd`)
- DTD (`dec` oder `dcl`)
- 

### **Hinweis für den XML Converter:**

Die Instanz muss als vollständiges Dokument in einer Datei abgebildet sein - Parameterentitäten bzw. Texteinschübe aus externen Dateien sind unzulässig.

Die Dokument-Typ-Deklaration (DTD) wird nicht ausgewertet. Es wird vorausgesetzt, dass die zu konvertierenden XML-/SGML-Instanzen entsprechend ihrer DTD valide sind.

Die DTD darf nicht im Dokument eingebettet sein.

## DTD

---

### Dokument-Typ-Deklaration.

Die DTD definiert den Aufbau eines Dokumentes. Hier werden die verwendeten Elemente, die Attribute der Elemente und die Inhaltsmodelle der Elemente definiert.

In der Praxis wird diese Deklaration als externe Datei gespeichert und verwaltet (Dateiname mit Endung `dtd`).

Eine DTD besteht aus einer beliebigen Kombination von:

- ELEMENT\* (Elementdefinitionen),
- ATTLIST\* (Attributdefinitionen) und
- ENTITY\* (Entitätendefinitionen).

### Hinweis (XML Converter):

Die DTD und deren Bestandteile werden nicht ausgewertet.

## **Einschluss**

---

### **(Inclusion)**

Einschlüsse sind zusätzlich zu einem Element definierte Elemente, die im Elementinhalt an beliebiger Stelle und in alle enthalten Unterelementen vorkommen dürfen.

### **Hinweis zum Auflösen von Einschlüssen/Ausschlüssen**

Folgende Prioritätsregeln sind bei der Bewertung der Gültigkeit von Ein- und Ausschlüssen unbedingt zu beachten.

- Ausschluss, wenn das zu untersuchende Element im Kontext als Ausschluss definiert wurde, darf es an der aktuellen Stelle nicht positioniert werden.
- Einschluss, wenn das zu untersuchende Element im Kontext als Einschluss definiert wurde, ist es an der aktuellen Stelle erlaubt.
- Element in der Modellgruppe, wenn das zu untersuchende Element weder als Ein- noch als Ausschluss im Kontext definiert wurde, wird die Gültigkeit ausschließlich durch die aktuelle Modellgruppe definiert (Definition in der DTD).

### **Hinweis**

Gilt nur für SGML.

### **Hinweis (XML Converter):**

Die DTD und deren Bestandteile werden nicht ausgewertet.

### **Siehe auch**

Ausschluss

## ELEMENT

---

Deklaration/Vereinbarung eines Elementes.

### Syntax XML:

**<!ELEMENT** *name* *inhaltsmodell* **>**

<i>name</i>	<i>Elementname</i>
<i>inhaltsmodell</i>	Definition des Inhaltsmodells: ANY beliebig erlaubter Inhalt EMPTY leeres Element oder einer expliziten Inhaltsbeschreibung bestehend aus: #PCDATA Textdaten/Zeichendaten Elementnamen zulässige Elemente im Inhalt Operatoren ",", " ", "(", ")", "?", "+"

### Syntax SGML:

**<!ELEMENT** *name* *sm* *em* *inhaltsmodell* *excl* *incl* **>**

**<!ELEMENT** *gruppe* *sm* *em* *inhaltsmodell* *excl* *incl* **>**

<i>name</i>	<i>Elementname</i>
<b>&lt;!ELEMENT</b> ( <i>gruppe</i> ) - o <b>ANY</b> <b>&gt;</b>	
<i>name</i>	Bezeichner für ein Element
<i>gruppe</i>	Elementgruppe. Ein oder mehrere Elementbezeichnungen, die mit Operatoren verbunden sind. Gültige Operatoren sind: & Und and Und   Oder or Oder , Sequenz Hinweis: Die Funktion der Operatoren wird in der Elementdeklaration ignoriert. Es werden nur die verwendeten Bezeichnungen der Elemente ausgewertet.
<i>sm</i>	Kennzeichen für Starttag-Minimierung: o Starttag kann ausgelassen werden - Starttag muss vorhanden sein.



<i>em</i>	Kennzeichen für Endtag-Minimierung <ul style="list-style-type: none"><li>o Endtag kann ausgelassen werden</li><li>- Endtag muss vorhanden sein.</li></ul>
<i>inhaltsmodell</i>	Definition des Inhaltsmodells: ANY beliebiger Inhalt EMPTY leeres Element CDATA reine Zeichendaten (Entitäten und Elemente werden nicht geparkt) RCDATA reine Zeichendaten und Entitäten (Elemente werden nicht geparkt) ANY beliebiger Inhalt erlaubt oder einer expliziten Inhaltsbeschreibung bestehen aus: #PCDATA Textdaten/Zeichendaten Elementnamen zulässige Elemente im Inhalt Operatoren ", "  ", "(" , ")", "?", "+"
<i>excl</i>	Ausschluss: - (Elementname oder Elementgruppe)
<i>incl</i>	Einschluss + (Elementname oder Elementgruppe)

**Hinweis (XML Converter):**

Die DTD und deren Bestandteile werden nicht ausgewertet.

## EMPTY

---

Schlüsselwort im Inhaltsmodell einer Elementdeklaration (siehe ELEMENT). Mit EMPTY werden wird im Inhaltsmodell eines Elementes dieses als leeres Element deklariert. Leere Elemente dürfen keinen Inhalt besitzen.

### Syntax XML:

```
<!ELEMENT name EMPTY >
```

*name* Bezeichner für ein Element

### Syntax SGML:

```
<!ELEMENT name - o EMPTY >
```

```
<!ELEMENT (gruppe) - o EMPTY >
```

*name* Bezeichner für ein Element

*gruppe* Elementgruppe. Ein oder mehrere Elementbezeichnungen, die mit Operatoren verbunden sind. Gültige Operatoren sind:

& Und

and Und

| Oder

or Oder

, Sequenz

Hinweis: Die Funktion der Operatoren wird in der Elementdeklaration ignoriert. Es werden nur die verwendeten Bezeichnungen der Elemente ausgewertet.

### Beispiel XML

Definition (DTD):

```
<!ELEMENT p (#PCDATA|index)* >  
<!ELEMENT b (#PCDATA|index)* >  
<!ELEMENT index EMPTY >  
<!ATTLIST index key CDATA REQUIRED >
```

Instanz (XML):

```
<p>Ein <b>Begriff<index key="Mein Stichwort"/></b>, der  
wichtig ist</p>
```

Im Beispiel wird das Element `p` und das Element `b` definiert, welche Zeichendaten und Stichwörter (`index`) enthalten dürfen. Das Element `index` ist ein leeres Element. Dieses Element besitzt keinen Endtag.

### Beispiel SGML

#### Definition (DTD):

```
<!ELEMENT p - - #PCDATA +index >
<!ELEMENT b - - #PCDATA >
<!ELEMENT index - o EMPTY >
<!ATTLIST index key CDATA REQUIRED >
```

#### Instanz (SGM):

```
<p>Ein <b>Begriff<index key="Mein Stichwort"></b>, der
wichtig ist</p>
```

Im Beispiel wird das Element `p` definiert, welches über den Einschluss (`+index`) im beliebigen Inhalt, wie auch in verschachtelten Elementen, das Element `index` enthalten darf. Das definierte Element `b` steht für die Hervorhebung fett. Das Element `index` ist ein leeres Element. Dieses Element besitzt keinen Endtag.

## ***Element***

---

Ein Element ist eine Daten- und Struktureinheit innerhalb von SGML/XML. In der SGML/XML-Syntax werden Elemente mit Hilfe so genannter "Tags" ausgezeichnet.

Abhängig vom definierten Inhaltsmodell werden Elemente mit Start- und Endtags (`<kapitel>...</kapitel>`) gekennzeichnet, oder bei leeren Elementen: a) mit nur einem Starttag (`<bild>`) in der SGML-Syntax bzw. b) mit einem Leertag (`<bild/>`) in der XML-Syntax.

Der *Elementname* ist hierbei der Bezeichner des Start- End- oder Leertags (in den Beispielen "kapitel", "bild").

Jedem Element kann entsprechend der DTD Eigenschaften zugeordnet werden, die als Attribute ausgezeichnet werden (`<kapitel id="BRR1">` bzw. `<bild file="bild1.jpg"/>`). Diese Eigenschaften werden mit den *Attributnamen* (hier "id" und "file") und den zugeordneten *Attributwerten* (hier "BRR1" und "bild1.jpg") gekennzeichnet.

### **Syntax 1 (Element mit Inhalt)**

`<Elementname{ Attribut}*>Elementinhalt</Elementname>`

### **Syntax 2 (leeres Element, SGML)**

`<Elementname{ Attribut}*>`

### **Syntax 3 (leeres Element, XML)**

`<Elementname{ Attribut}*!>`

Leeres Element.

`<...>` Anfangsmarke eines *Elementes* (Starttag)

`</...>` Endmarke eines *Elementes* (Endtag)

`<.../>` Marke eines leeren *Elementes* (nur XML)

*Elementname* *Bezeichner* eines *Elementes*

*Attribut* *Eigenschaften* eines *Elementes*

**Elementname**

---

Ein Elementname ist ein gültiger *Bezeichner* für ein *Element*.

Z.B. in XML-/SGML-Dokumenten:

**Syntax 1 (Element mit Inhalt)**

`<Elementname{ Attribut}*>Elementinhalt</Elementname>`

**Syntax 2 (leeres Element, SGML)**

`<Elementname{ Attribut}*>`

**Syntax 3 (leeres Element, XML)**

`<Elementname{ Attribut}* />`

Leeres Element.

**Typ**

*Dokument*

**Siehe auch**

*Skript*

## ENTITY

---

Deklaration/Vereinbarung einer Entität (Dateneinheit).

Achtung, hier wir nur der Typ der allgemeinen Entität (**general entity**) beschrieben. Der Typ Parameterentität (**parameter entity**) wird vom Skript nicht unterstützt und hier nicht weiter beschrieben (zu erkennen sind aber Parameterentitäten am "% " vor dem Bezeichner).

### Syntax XML 1

**<!ENTITY** *Bezeichner Text***>**

### Syntax XML 2

**<!ENTITY** *Bezeichner SYSTEM System-ID***>**

### Syntax XML 3

**<!ENTITY** *Bezeichner SYSTEM System-ID Typ***>**

### Syntax XML 4

**<!ENTITY** *Bezeichner PUBLIC Public-ID***>**

### Syntax XML 5

**<!ENTITY** *Bezeichner PUBLIC Public-ID System-ID***>**

### Syntax XML 6

**<!ENTITY** *Bezeichner PUBLIC Public-ID System-ID Typ***>**

<i>Bezeichner</i>	Name der Entität ( <i>Bezeichner</i> )
<i>Text</i>	<i>Zeichenkette</i> , des definierten Textes.
<i>System-ID</i>	<i>Zeichenkette</i> , die einen Systembezeichner enthält
<i>Public-ID</i>	<i>Zeichenkette</i> , die einen öffentlichen Bezeichner enthält
<i>Typ</i>	Entitätentyp: <b>NDATA</b> {Notation} Die Notation muss mit NOTATION deklariert sein!

**Syntax SGML 1****<!ENTITY** *Bezeichner Text***>****Syntax SGML 2****<!ENTITY** *Bezeichner CDATA Text***>****Syntax SGML 3****<!ENTITY** *Bezeichner SDATA Text***>****Syntax SGML 4****<!ENTITY** *Bezeichner PI Text***>****Syntax SGML 5****<!ENTITY** *Bezeichner SYSTEM System-ID***>****Syntax SGML 6****<!ENTITY** *Bezeichner SYSTEM System-ID Typ***>****Syntax SGML 7****<!ENTITY** *Bezeichner PUBLIC Public-ID***>****Syntax SGML 8****<!ENTITY** *Bezeichner PUBLIC Public-ID System-ID***>****Syntax SGML 9****<!ENTITY** *Bezeichner PUBLIC Public-ID System-ID Typ***>**

<i>Bezeichner</i>	Name der Entität ( <i>Bezeichner</i> )
<i>Text</i>	<i>Zeichenkette</i> , des definierten Textes.
<i>System-ID</i>	<i>Zeichenkette</i> , die einen Systembezeichner enthält
<i>Public-ID</i>	<i>Zeichenkette</i> , die einen öffentlichen Bezeichner enthält
<i>Typ</i>	Entitätentyp: SUBDOC NDATA { <i>Notation</i> } CDATA { <i>Notation</i> } SDATA { <i>Notation</i> } Die Notation muss mit NOTATION deklariert sein!

## Syntax für eine interne Dateneinheit

**<!ENTITY** *Bezeichner Text***>**

*Bezeichner*

Name der Entität (*Bezeichner*)

*Text*

*Zeichenkette*, des definierten Textes.

### Beispiel

```
<!ENTITY auml "ae">
```

In diesem Beispiel wird die Dateneinheit `auuml` definiert, dessen Ersetzungstext `"ae"` ist.

## Syntax für externe Grafiken

**<!ENTITY** *Bezeichner* **SYSTEM** *System-ID* **NDATA** *Notation***>**

*Bezeichner*

Name der Entität (*Bezeichner*)

*System-ID*

*Zeichenkette*, die einen Systembezeichner enthält (Dateiname)

*Notation*

Typ der externen Datei:

**TIFF**

**EPSI**

Die Notation muss in der DTD definiert sein (Siehe NOTATION)!

### Beispiel

```
<!DOCTYPE dokument PUBLIC "-//Seume//DTD Handbuch  
V1.00//DE" [  
<!ENTITY entity.001 SYSTEM "grafik.tif" NDATA bmp>  
]>
```

In dieser erweiterten Deklaration wird die externe Grafikdatei `"grafik.tif"` als Entität `"entity.001"` vom Typ `NDATA` deklariert.

### Hinweis (XML Converter):

Die DTD und deren Bestandteile werden nicht ausgewertet.

## **Entität**

---

Eine Entität ist eine Dateneinheit.

Diese besteht aus der Definition der Dateneinheit (ENTITY) innerhalb der DTD oder der erweiterten Vereinbarung (Vereinbarung, erweiterte) im Prolog des Dokumentes und dem Referenzieren dieser Dateneinheit innerhalb des Dokumentes.

Die Definition der Dateneinheit erfolgt mit der Vereinbarung ENTITY und ist dort näher beschrieben.

Hier folgt die Beschreibung der Referenz.

In XML und SGML sind beliebige Dateneinheiten über Entitäten definierbar. Im Skript und den verwendeten Daten sollten nur Einheiten vom Typ Zeichenverweis verwendet werden.

### **Syntax**

**&Entitätenname;**

<b>&amp;</b>	Beginn der Entität
<i>Entitätenname</i>	<i>Bezeichner</i> der Entität
<b>;</b>	Ende der Entität

Aufruf der Dateneinheit mit dem angegebenen Bezeichner (Entitätenname). Der Inhalt der Dateneinheit wird anstelle dieser Referenz (Entität) in das Dokument eingefügt.

### **Beispiel für Zeichenverweis**

```
&auml;
```

Typische Kennzeichnung des deutschen Umlautes "ü" innerhalb von SGML-Dokumenten. Innerhalb von XML-Dokumenten ist diese Darstellung eher untypisch, wird aber zur Verbesserung der Lesbarkeit trotzdem eingesetzt (denn die gleichbedeutende numerische Referenz `&#xFC;` oder `&#252;` erschwert das Lesen).

### **Siehe auch**

ENTITY, Zeichenverweis

## Kommentar

---

Datenbereiche, die vom Parser unberücksichtigt gelassen werden sollen, z.B. zum Einfügen von Anmerkungen.

### Syntax 1

`<!-- Text -->`

`<!--` Beginn des Kommentars

`-->` Ende des Kommentars

*Text* beliebiger Text mit Ausnahme der Kommentar-Ende-Kennung (`-->`).

Kommentarvereinbarung. Syntaktisch besteht dieser Kommentar aus einer leeren Vereinbarung (`<!>`), die einen Kommentar (`<!-- . . . -->`) entsprechend Syntax 2 (SGML) enthält.

### Syntax 2 (nur SGML)

`-- Text --`

`--` Beginn des Kommentars

`--` Ende des Kommentars

*Text* beliebiger Text mit Ausnahme der Kommentar-Ende-Kennung (`--`).

Gilt nur für SGML und darf nur innerhalb von Vereinbarungen verwendet werden.

### Hinweis (XML Converter):

Kommentare werden nicht ausgewertet.

***Entitätenname***

---

Ein Entitätenname ist ein gültiger *Bezeichner* eine benannte Entität.

**Instanz**

---

Der eigentliche Dokumentinhalt (Daten) des Dokumentes (*Dokument*).

Dieser Dateninhalt besteht aus *Elementen* beginnend mit dem im DOCTYPE definierten Basiselement.

## Leerraum

---

(Whitespace)

Ein Leerraum ist ein Zeichen oder ein Zeichenbereich bestehend aus:

- Leerzeichen " ",
- CR (13),
- LF (10) und
- Tabulator (9).

Leerräume werden in der Syntax von XML-/SGML-Daten zur Abgrenzung der syntaktischen Bestandteile verwendet. Beispiele:

```
<!DOCTYPE{Leerraum}dokument{Leerraum}  
SYSTEM{Leerraum}"dokument.dtd">{Leerraum}...
```

oder

```
<dokument{Leerraum}id="dr1255">{Leerraum}
```

oder

```
<dokument>{Leerraum}<kapitel>{Leerraum}<absatz>Text Text</  
absatz>{Leerraum}
```

Innerhalb dieses Handbuches werden Leerräume nur als " " in der Syntax dargestellt. Natürlich können statt des einfachen Leerzeichens alle Zeichen entsprechend der Definition des Leerraumes verwendet werden.

### Hinweis

Leerräume in XML-/SGML-Daten außerhalb von Inhaltsbereichen, die als PCDATA deklariert sind, werden vom Parser nicht als Daten gelesen. Diese Leerräume werden lediglich zur "Formatierung" der Dokumente verwendet, also zur zeilenweisen Gestaltung oder Einrückung. Diese Leerräume sind keine Daten!

## NDATA

---

(non **data**, non text-**data**)

Nicht XML-/SGML-Daten, also alle externen Binärdaten. In der Regel werden externe Grafiken als NDATA definiert und referenziert.

### Beispiel

```
<!DOCTYPE dokument PUBLIC "-//Seume//DTD Handbuch
V1.00//DE" [
<!ENTITY entity.001 SYSTEM "grafik.tif" NDATA bmp>
]>
```

In dieser erweiterten Deklaration wird die externe Grafikdatei "grafik.tif" als Entität "entity.001" vom Typ NDATA deklariert.

## **NAMECASE**

---

Beachtung der Groß- und Kleinschreibung von Elementnamen, Attributnamen, Entitätennamen, usw..

### **Hinweis (XML Converter):**

Folgende Einstellungen werden vorausgesetzt:

### **XML**

#### **NAMECASE**

GENERAL=**NO**

ENTITY=NO

Die Groß/Kleinschreibung aller Bezeichner (Elementnamen, Attributnamen, Entitätennamen, usw.) muss gemäß DTD eingehalten werden.

### **SGML**

#### **NAMECASE**

GENERAL=**YES**

ENTITY=NO

Die Groß/Kleinschreibung der Bezeichner von Elementnamen und Attributnamen kann beliebig erfolgen, die Groß/Kleinschreibung der Bezeichner von Entitätennamen, usw.) muss gemäß DTD eingehalten werden.

### **Hinweis**

Nur in **SGML** verfügbar!

### **Hinweis (XML Converter):**

Die Einstellung muss mit dem Befehl NAMECASE vorgenommen werden, wenn diese von der Voreinstellung (SGML = Groß-/Kleinschreibung ignorieren und XML = Groß-/Kleinschreibung beachten) abweichen soll.

### **Typ**

Deklaration

## NOTATION

---

Definition der Verarbeitungsart von Daten durch ein externes Programm.

### Syntax 1

**<!NOTATION** *Bezeichner* SYSTEM *System-ID*>

*Bezeichner*                      Name der Notation.

*System-ID*                      Dateiname des Programms.

Siehe auch SYSTEM.

### Syntax 2

**<!NOTATION** *Bezeichner* PUBLIC *Public-ID*>

### Syntax 3

**<!NOTATION** *Bezeichner* PUBLIC *Public-ID* *System-ID*>

*Bezeichner*                      Name der Notation.

*Public-ID*                      Öffentlicher Bezeichner für das Programm.

*System-ID*                      Dateiname des Programms.

Siehe auch PUBLIC.

### Beispiel

```
<!NOTATION TIFF SYSTEM "bild/tiff">
```

### Hinweis (XML Converter):

Die DTD und deren Bestandteile werden nicht ausgewertet.

## OMITTAG

---

Auslassung von Start- und Endtags.

Es wird

OMITTAG=NO

vorausgesetzt.

Unter der Bedingung, dass ausschließlich bei leeren Elementen der Endtag weggelassen wird und dieses im Skript zum entsprechenden Element (MODEL) definiert wird, können auch Daten mit

OMITTAG=YES

verarbeitet werden.

### Beschreibung

In der folgenden Beschreibung wird die OMIT-Definitionen der SGML-DTD verwendet: (o = *omit*, auslassen erlaubt, - = auslassen verboten; der erste Wert steht für den Starttag, der zweite Wert steht für den Endtag):

- o o Sowohl der Start- als auch der Endtag können ausgelassen werden. In den Daten taucht in diesem Fall kein Tag auf. Der Parser liest aber entsprechend der DTD das betreffende Element! Nicht verwenden!
- o - Der Starttag kann ausgelassen werden, der Endtag muss eingetragen werden. Diese Kombination ist rein theoretisch. Nicht verwenden!
- o Der Starttag muss angegeben werden, der Endtag kann ausgelassen werden. Dieser Fall wird typischerweise in leeren Elementen verwendet. Im Skript müssen solche Elemente mit MODEL=EMPTY gekennzeichnet werden
- - Start- und Endtag müssen angegeben werden.

### Hinweis

Nur in **SGML** verfügbar!

### Hinweis

Vom Skript nicht unterstützt!

### Typ

Deklaration

## **PCDATA**

---

Schlüsselwort im Inhaltsmodell einer Elementdeklaration (siehe ELEMENT).

### **Parsed Character DATA**

Zeichendaten, die Zeichen und *Entitäten* enthalten können (alles außer Tags).

Schreibweise:

### **#PCDATA**

Alle Zeilenumbrüche und Tabulatoren werden als Informationen gewertet!

### **Hinweis**

Wenn die Daten konvertiert werden sollen, deren DTD unbekannt ist, so sollten Zeilenumbrüche und Tabulatoren im PCDATA-Bereich vermieden werden. Diese Bereiche erkennt man relativ einfach daran, dass es Elemente sind, die Textinformationen enthalten. Elemente, die lediglich Unterelemente enthalten und keine Textinformationen können mit Tabulatoren, Leerzeichen und Zeilenumbrüchen versehen werden, ohne dass es zu inhaltlichen Verfälschungen kommt!

## Prolog

---

Der Prolog enthält alle Definitionen für den folgenden Dokumentinhalt. So wird werden hier über die XML-Deklaration:

- die XML-Version,
- die Kodierung,
- die Existenz einer DTD

und über den DOCTYPE:

- das Basiselement (Hauptelement des Dokumentes),
- der öffentliche Bezeichner (PUBLIC) und
- die verwendete DTD (SYSTEM)

definiert.

### Syntax XML

XML-Deklaration?, DOCTYPE?

### Syntax SGML

DOCTYPE?

### Hinweis zu XML

Bei XML-Dokumenten wird dem Prolog die XML-Deklaration als Ausführungsanweisung vorangestellt:

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

## **PUBLIC**

---

Öffentlicher Bezeichner (ID).

### Syntax

**PUBLIC** *Public-ID System-ID*

*Public-ID* eingeschränkte *Zeichenkette*, die einen öffentlichen Bezeichner enthält (siehe *Zeichenkette*, eingeschränkte)

*System-ID* eingeschränkte *Zeichenkette*, die einen Systembezeichner (siehe *Zeichenkette*, eingeschränkte und SYSTEM) enthält

### Syntax des Bezeichners

**"Norm//Besitzer//Klasse Spezifikation//Sprache"**

*Norm* Ist dieser Bezeichner als Norm angemeldet:

- + ISO angemeldet
- nicht angemeldet

*Besitzer* Kennzeichnung des Besitzers bzw. Firmenname

*Klasse* Was wird definiert?

**DTD** öffentlicher Bezeichner für eine DTD

*Spezifikation* Der eigentliche öffentliche Bezeichner einschließlich der Versionskennzeichnung

*Sprache* Sprache:

- DE** deutsch
- EN** englisch
- ES** spanisch
- FR** französisch

### Beispiel

```
<!DOCTYPE dokument PUBLIC "-//Seume//DTD Handbuch  
V1.00//DE">
```

**RCDATA**

---

(replaceable **character data**)

Datenbereiche, die reine Textinformationen wie CDATA und *Entitäten* enthalten dürfen. Elemente sind nicht erlaubt.

**Hinweis**

Nur in **SGML** verfügbar!

## **SDATA**

---

(system specific **data**)

Systemabhängige Daten. In der Regel werden alle außerhalb des SGML-Zeichenbereiches liegenden Zeichen, als SDATA Entitäten (siehe ENTITY) definiert. Eine Angabe der Kodierung erfolgt hierbei nicht. Es wird dem System überlassen, das korrekte Zeichen darzustellen.

### **Beispiel**

```
<!ENTITY auml SDATA "[auuml]">
```

Der deutsche Umlaut zu a wird als benannte Entität `auuml` definiert. Es ist dem System überlassen, das passende Zeichen (ä) dafür zu ersetzen.

### **Hinweis**

Nur in **SGML** verfügbar!

### **Hinweis (XML Converter):**

Die DTD und deren Bestandteile werden nicht ausgewertet.

## SGML-Deklaration

---

Nach Norm leitet die SGML-Deklaration ein SGML-Dokument ein.

In der Praxis wird diese Deklaration jedoch als externe Datei gespeichert und verwaltet (Dateiname mit Endung `dcl` oder `dec`).

### Syntax

**<!SGML** *version*

**CHARSET** *zeichensatz*

**CAPACITY** *kapazitäten*

**SCOPE** *bereich*

**SYNTAX** *syntax*

**FEATURES** *eigenschaften*

**APPINFO** *info*

>

<i>version</i>	Zeichenkette mit der verwendeten Version der SGML-Norm. Zur Zeit (und immer noch) gültiger Wert: "ISO 8879:1986"
<i>zeichensatz</i>	Definiert die gültigen Zeichen für alle Bestandteile eines SGML-Dokumentes. {BASESET, DESCSET}+
<i>kapazitäten</i>	Definiert
<i>bereich</i>	Definitionen gelten für: DOCUMENT oder INSTANCE
<i>syntax</i>	Ausführliche Definition der SGML-Strukturbestandteile (Syntax für Elemente, Attribute, Entitäten) in Qualität und Quantität. Wichtig ist im Abschnitt NAMING die Definition der möglichen Groß-/Kleinschreibung der Elemente und Entitäten NAMECASE.
<i>eigenschaften</i>	benutzte erweiterte SGML-Eigenschaften: MINIMIZE {DATATAG, OMITTAG, RANK, SHORT-TAG} LINK {SIMPLE, IMPLICIT, EXPLICIT} OTHER {CONCUR, SUBDOC, FORMAL}



*info*

Zusätzliche Anwendungsinformationen oder NONE.

**Hinweis (XML Converter):**

Die SGML-Deklaration wird nicht unterstützt.

**Siehe auch**

[SGML]

## SHORTTAG

---

Definition von verkürzten Schreibweisen von Elementen und Attributen.

Es wird

SHORTTAG=NO

vorausgesetzt.

### Beschreibung

Folgende Syntaxerweiterungen werden durch SHORTTAG ausgelöst:

<>	leerer Starttag Beginnt ein Element, welches durch Kontext und DTD eindeutig erkannt werden kann. Der Elementname kann in diesem Fall ausgelassen werden.
</>	leerer Endtag Schließt das offene Element entsprechend Kontext/ DTD.
<tag1<tag2>	nichtgeschlossene (verkürzte) Tags. Ein Tag wird auch dadurch geschlossen (>), dass ein neuer Tag (<) begonnen wird.
<tag1 id=wert1>	Attributwerte, die nicht in Anführungszeichen bzw. Hochkommas gesetzt werden.
<tag1 wert1>	Attributnamen, die entsprechend der DTD eindeutig erkannt werden können, können ausgelassen werden. Es genügt für diesen Fall die Angabe des Attributwertes.

Diese Erweiterungen sind hier **nicht erlaubt!**

### Hinweis

Wenn Sie eine DTD/Deklaration mit SHORTTAG=YES verwenden wollen, müssen Sie diese DTD/Deklaration auf SHORTTAG=NO umstellen und gegebenenfalls die Daten umarbeiten.

### Hinweis

Nur in **SGML** verfügbar!

### Hinweis

Vom Skript nicht unterstützt!

### Typ

Deklaration



## SUBDOC

---

Mit dem Entitätentyp SUBDOC können externe Unterdokumente über eingebunden werden. Dazu muss im Dokument eine Entität (siehe ENTITY) vom Typ SUBDOC für das externe *Dokument* definiert werden. Dieses externe Dokument muss als eigenständiges Dokument betrachtet werden, d.h., es hat eine eigene DTD, die von der DTD des aufrufenden Dokumentes abweichen kann.

### Beispiel:

```
<!DOCTYPE buch PUBLIC "-//Seume//DTD Handbuch V1.00//DE"
[
<!ENTITY kap1 SYSTEM "kapitel1.sgm" SUBDOC>
<!ENTITY kap2 SYSTEM "kapitel1.sgm" SUBDOC>
<!ENTITY anhang SYSTEM "anhang.sgm" SUBDOC>
]>
<buch>
&kap1;
&kap2;
&anhang;
</buch>
```

Dokument mit dem Hauptelement "buch" besteht aus den externen Dokumenten "kap1.sgm", "kap2.sgm" und "anhang.sgm".

### Hinweis zu XML

Da SUBDOC in XML nicht verfügbar ist, muss dort für externe Dokumente auf allgemeine Entitäten zurückgegriffen werden. Es ist dabei allerdings zu beachten, dass alle Bestandteile auf Grundlage der gleichen DTD definiert sein müssen. Weiterhin muss beim Einfügen der Inhalte der externen Entitäten ein gültiges (valides) Inhaltsmodell des Hauptdokumentes entstehen!

### Hinweis (XML Converter):

Nicht unterstützt.

### Hinweis

Nur in **SGML** verfügbar!

**SYSTEM**

---

Systembezeichner, in der Regel eine Dateibezeichnung.

**Syntax 1****SYSTEM** *System-ID*

*System-ID* eingeschränkte *Zeichenkette*, die einen Systembezeichner enthält (siehe *Zeichenkette*, eingeschränkte).

**Syntax 2****PUBLIC** *Public-ID* *System-ID*

*Public-ID* eingeschränkte *Zeichenkette*, die einen öffentlichen Bezeichner (siehe auch PUBLIC) enthält.

*System-ID* eingeschränkte *Zeichenkette*, die einen Systembezeichner enthält (siehe *Zeichenkette*, eingeschränkte). Innerhalb von XML ist der System-ID eine Pflichtangabe, innerhalb von SGML ist diese Angabe optional.



## **UCS-2**

---

16 Bit Zeichenkodierung.

Jedes Zeichen wird durch 2 Byte umschrieben.

Der Binärwert entspricht hierbei exakt dem Zeichencode.

UCS-2 entspricht UTF-16 Version 1.0.

UCS.2 beschreibt den Zeichenbereich Unicode Plane 0 (Basic Multilingual Plane, BMP), also die ersten 64 K Zeichen von Unicode.

### **Zeichenbereich**

x0 bis xFFFF

### **Hinweis**

UCS-2 beschreibt nur die BMP von UTF-16 Version 2.0 ff.

UCS-2 ist aufwärtskompatibel zu UTF-16 Version 2.0 ff.

UTF-16 Version 2.0 ff. ist nicht abwärtskompatibel zu UCS-2.

### **Siehe**

[UCS-2], [UNICODE]

**UCS-4**

---

32 Bit Zeichenkodierung.

Jedes Zeichen wird durch 4 Bytes umschrieben.

Der Binärwert entspricht hierbei exakt dem Zeichencode.

USC-4 entspricht UTF-32.

**Zeichenbereich**

x0 bis x10FFFF (nach Unicode-Norm)

**Siehe**

[UNICODE]

**UTF-8**

---

8 Bit Zeichenkodierung mit variabler Länge.  
Jedes Zeichen wird durch 1 Byte bis 4 Bytes.

**Zeichenbereich**

x0 bis x10FFFF

**Siehe**

[UNICODE]

**UTF-16**

---

16 Bit Zeichenkodierung mit variabler Länge.

Jedes Zeichen wird durch 2 Bytes oder 4 Bytes umschrieben.

**Zeichenbereich**

x0 bis x10FFFF

**Hinweis**

In der Version Unicode 1.x (1991-1996) war UTF-16 nur für 64 K Zeichen definiert. Dieser Zeichenbereich entspricht ab der Version Unicode 2.x der Unicode Plane 0 (Basic Multilingual Plane, BMP). Unicode 1.x UTF-16 ist identisch zu UCS-2. Jedes Zeichen wird in der Unicode Version 1.x durch exakt 2 Bytes umschrieben.

**Siehe**

[UNICODE]

**UTF-32**

---

32 Bit Zeichenkodierung.

Jedes Zeichen wird durch 4 Bytes umschrieben.

Der Binärwert entspricht hierbei exakt dem Zeichencode.

**Zeichenbereich**

x0 bis x10FFFF

**Siehe**

[UNICODE]

## Vereinbarung

---

Vereinbarung (Deklaration) eines Syntaxbestandteils des Dokumentes.

### Syntax

*<!Typ Parameter>*

< Start der Vereinbarung

> Ende der Vereinbarung

*Typ* Typ der Vereinbarung, folgende Werte sind möglich:

XML XML-Deklaration

SGML SGML-Deklaration

DOCTYPE

ELEMENT

ENTITY

{Kommentar}

*Parameter* Abhängig vom Typ (siehe auch deren Beschreibungen)

## Vereinbarung, erweiterte

---

Erweiterte Vereinbarung.

Dient zum Definieren zusätzlicher Bestandteile zusätzlich zur DTD.

Eine erweiterte Vereinbarung ist nur im DOCTYPE erlaubt.

### Syntax

**[{*Vereinbarung*}\*]**

[

Beginn der erweiterten Vereinbarung

]

Ende der erweiterten Vereinbarung

*Vereinbarung*

zusätzliche Vereinbarungen, ausgenommen  
DOCTYPE.

### Beispiel

```
<!DOCTYPE dokument PUBLIC "-//Seume//DTD Handbuch  
V1.00//DE" [  
<!ENTITY entity.001 SYSTEM "grafik.tif" NDATA bmp>  
]>
```

In der erweiterten Deklaration wird die zusätzliche externe Grafikdatei "grafik.tif" als Entität "entity.001" deklariert.



**Vereinbarung, leere**

---

Leere Vereinbarung.



## Syntax

<!\>

## XML-Deklaration

---

Die XML-Deklaration leitet ein XML-Dokument ein. Diese Deklaration wird empfohlen. Der Einsatz ist jedoch nicht zwingend erforderlich.

Wichtigster Bestandteil der XML-Deklaration ist die Angabe der Kodierung. Wenn ein Dokument ein BOM enthält, muss die Kodierung im BOM und in der XML-Deklaration übereinstimmen. Es empfiehlt sich, die Kodierung innerhalb der XML-Deklaration stets anzugeben, da der BOM in der Regel in den gängigen Editoren nicht angezeigt wird.

### Syntax

**<?xml version encoding? standalone? ?>**

<i>version</i>	Version des XML-Dokumentes in Schreibweise eines Attributes: <code>version="zeichenkette"</code> Zur Zeit gültige Werte der XML-Version: 1.0      XML Version 1.0 Siehe auch [XML].				
<i>encoding</i>	Kodierung des XML-Dokumentes in Schreibweise eines Attributes <code>encoding="zeichenkette"</code> Gültige Werte für die Kodierung sind z.B.: ASCII            ASCII-Kodierung UTF-8            UTF-8-Kodierung UTF-16           UTF-16-Kodierung UTF-32           UTF-32-Kodierung ... Siehe auch [XML].				
<i>standalone</i>	Definiert, ob das Dokument ohne externe Definitionen (DTD) gültig ist. Diese Angabe wird in Schreibweise eines Attributes angegeben: <code>encoding="zeichenkette"</code> Gültige Werte sind: <table><tr><td><i>yes</i></td><td>Dokument ist "alleinstehend" gültig (keine DTD bzw. keine externe Definitionen erforderlich).</td></tr><tr><td><i>no</i></td><td>Dokument kann externe Definitionen nutzen.</td></tr></table>	<i>yes</i>	Dokument ist "alleinstehend" gültig (keine DTD bzw. keine externe Definitionen erforderlich).	<i>no</i>	Dokument kann externe Definitionen nutzen.
<i>yes</i>	Dokument ist "alleinstehend" gültig (keine DTD bzw. keine externe Definitionen erforderlich).				
<i>no</i>	Dokument kann externe Definitionen nutzen.				

### Beispiele 1

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

**Beispiele 2**

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

**Beispiele 3**

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
```

Siehe auch

[XML]

## Zeichenverweis

---

Referenz, die ein Zeichen darstellt.

### Syntax 1

**&#Dezimalzahl;**

*Dezimalzahl*                      Numerischer Zeichencode für das Zeichen, erlaubte  
Dezimalzeichen sind  
0123456789

### Syntax 2 (XML und SGML gemäß Web SGML Adaption)

**&#x{Hexadezimalzahl};**

*Hexadezimalzahl*                      Numerischer Zeichencode für das Zeichen, erlaubte  
Hexadezimalzeichen sind:  
0123456789ABCDEF

Im SGML-Standard (siehe [SGML]) nicht enthalten, aber in der Erweiterung für das Web (siehe [Web SGML]) definiert.

### Syntax 3 (SGML)

**&#Funktion;**

*Funktion*                                      *Bezeichner* für das Zeichen, gültige vordefinierte  
Werte sind:  
**RE**                      (13) Wagenrücklauf  
**RS**                      (10) Zeilenvorschub  
**TAB**                      (9) Tabulator  
**SPACE**                      (32) Leerzeichen

Funktionszeichen. Die definierten Zeichen werden im Abschnitt `FUNCTION` innerhalb SGML-Deklaration der definiert.



# Editoren

## Editoren mit integrierter Unterstützung für XML-Converter

### PSPad Editor Version 5.0.3 (377)

Wenn Sie PSPad Version 4.6.0 vom 2.10.2015 oder neuer bereits installiert haben, können Sie die Installationsanleitung überspringen. Die folgende Anleitung ist für eine Nachinstallation im Problemfall vorgesehen.

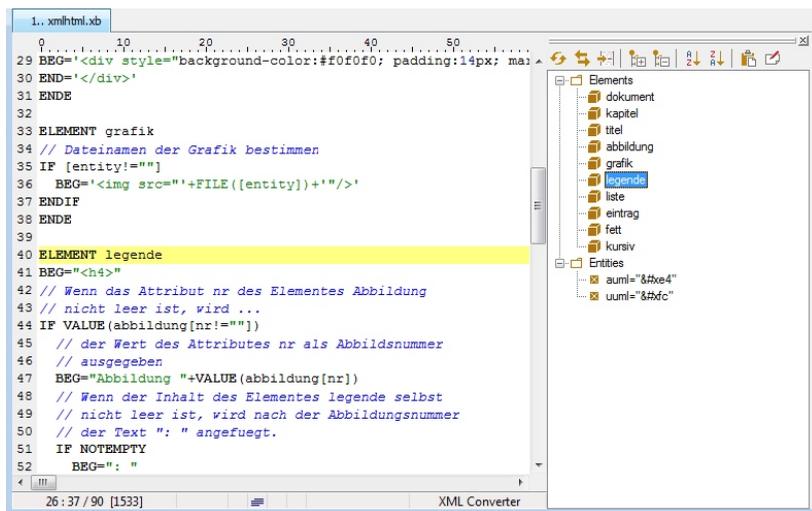
### Syntaxhervorhebungen

Alle Skript-Dateien mit der Dateieindung ". x.b" werden den Highlighter-Einstellungen „XB“ automatisch zugeordnet.

### Code-Navigation

Öffnen Sie den Code-Explorer über das Menü "Werkzeuge / Code-Explorer einblenden ...". Alle Skript-Dateien mit der Dateieindung ". x.b" werden automatisch analysiert und im Code-Explorer zur schnellen Navigation in Gruppen Elements und Entities angezeigt.

### Bildschirmfoto





## Debuggen

Verwenden Sie die Compiler-Einstellungen des Highlighters oder eines Projektes zum Debuggen von Konvertierungsprozessen. Stellen Sie den „Log-Parser“ dazu wie folgt ein:

```
in Datei '%F' Zeile=%L, Zeichen=%C
```

Als Compiler tragen Sie zum Beispiel die Batchdatei Ihres Projektes ein:

```
C:\Programme\XML Converter\Projekt\Test.bat
```

und das Startverzeichnis:

```
C:\Programme\XML Converter\Projekt
```

und die Logdatei, die in der Batchdatei als Ausgabe definiert wurde:

```
C:\Programme\XML Converter\Projekt\Test.log
```

Wenn Sie die Skriptdatei:

```
C:\Programme\XML Converter\Projekt\xmlhtml.xb
```

mit dem Editor bearbeiten, können Sie mit „Strg+F9“ die Konvertierung starten und erhalten die Logausgabe im Logfenster des Editors angezeigt. Mit einem Klick auf eine Fehlermeldung, die auf eine Dateiposition verweist, wird die angegebene Datei geöffnet und die Position markiert. Wenn Sie ein Projekt definiert haben, wird für alle Dateien des Projektes mit „Strg+F9“ die Konvertierung gestartet und das Logausgabe ausgewertet.

## Siehe

[PSPad]



---

## Konfigurationsanleitungen

---

### BBEdit

---

#### Kompatibilität

macOS

#### Konfigurationsdateien

```
Editoren\BBEdit\xml_converter.plist
```

#### Hinweis

Diese Konfiguration wurde für den **BBEdit 11.6.2** erstellt.

#### Installation

- Kopieren Sie die Konfigurationsdatei (`xml_converter.plist`) in den Ordner `~/Library/Application Support/BBEdit/Language Modules/`. Kopieren Sie dazu die Konfigurationsdatei in die Zwischenablage, wählen Sie im Finder `Gehe zu`, halten Sie die `Alt`-Taste gedrückt, so dass in der Liste `Library` erscheint und wählen Sie `Library` aus. Jetzt wählen Sie `Application Support/BBEdit/Language Modules/`. Fügen Sie den Inhalt der Zwischenablage hier ein.

#### Hinweis

Der Benutzerordner in macOS wird mit `~` angegeben. Sie können auch `user/name` (für `name` bitte den Benutzerordnernamen angeben) verwenden.

#### Syntaxhervorhebungen

Alle Skript-Dateien mit der Dateierdung `.xb` werden den Highlighter-Einstellungen `XB` automatisch zugeordnet.

#### Siehe

[BBEdit]



## Brackets

---

### Kompatibilität

Linux, macOS, Windows

### Konfigurationsdateien

Editoren\Nackets\brackets-xb-syntax.zip

### Hinweis

Diese Konfiguration wurde für den **Brackets 1.3** erstellt.

### Installation

- Starten Sie Brackets.
- Wählen Sie im Menü `Datei/Erweiterungs-Verwaltung...`
- Im unteren Bereich der Erweiterungs-Verwaltung befindet sich ein Steuerelement  
`.zip hierhinziehen oder von URL installieren...`  
Ziehen Sie die Konfigurationsdatei `brackets-xb-syntax.zip` auf diese Schaltfläche.
- Schließen Sie die Erweiterungsverwaltung und starten Sie Brackets neu.

### Syntaxhervorhebungen

Alle Skript-Dateien mit der Dateiendung `.xb` werden den Highlighter-Einstellungen `XB` automatisch zugeordnet.

### Siehe

[Brackets]



## CodeMirror

---

### Kompatibilität

Browser (Linux, macOS, Windows)

### Konfigurationsdateien

Editoren\CodeMirror\xb.zip

### Hinweis

Diese Konfiguration wurde für den **CodeMirror 5.3** erstellt.

### Installation

- Extrahieren Sie die Konfigurationsdateien `xb.zip`.
- Kopieren Sie den Ordner `xb` in den CodeMirror `mode` Ordner.

### Siehe

[CodeMirror]



## Codepad 2

---

### Kompatibilität

Windows

### Konfigurationsdateien

Editoren\CodePad2\XB.xml

### Hinweis

Diese Konfiguration wurde für **Codepad Version 2.0.4.0** (deutsche Version) erstellt.

### Installation

- Kopieren Sie die Syntax-Konfigurationsdatei (.xml) in den Highlighter-Ordner von Codepad. Dieser befindet sich im Programmverzeichnis (in Windows x64 C:\Program Files (x86) andernfalls C:\Programme) im Unterordner (Codepad\profile\Highlighter).
- Starten Sie Codepad.

### Siehe

[Codepad2]



## CodePad 4

---

### Kompatibilität

Windows

### Konfigurationsdateien

```
Editoren\CodePad4\xb.def
```

### Hinweis

Diese Konfiguration wurde für **CodePad Version 4.1** (deutsche Version) erstellt.

### Installation

- Kopieren Sie die Syntax-Konfigurationsdatei (`.def`) in den Syntaxordner von CodePad. Dieser befindet sich im Programmverzeichnis (in Windows x64 `C:\Program Files (x86)` andernfalls `C:\Programme`) im Unterordner (`CodePad\Syntax Definitions`).
- Starten Sie CodePad.

### Debuggen

Verwenden Sie die Benutzeraktionen zum Debuggen von Konvertierungsprozessen. Stellen Sie den „Compilerauschrift Parser Regel“ dazu wie folgt ein:

```
in Datei '%n' Zeile=%l, Zeichen=%c.
```

Als „Befehl“ für die Benutzeraktion tragen Sie zum Beispiel die Batchdatei Ihres Projektes ein:

```
C:\Programme\XML Converter\Projekt\Test.bat
```

und geben das Startverzeichnis in „Start in“ an:

```
C:\Programme\XML Converter\Projekt
```

Beachten Sie, dass die Ausgabe im Batchfile nicht in eine Datei umgeleitet werden darf. Die Benutzeraktion „fängt die Ausgabe“ auf.

(ungetestet)

### Siehe

[CodePad4]



## Fraise

---

### Kompatibilität

macOS

### Konfigurationsdateien

Editoren/Fraise/xmlconverter.plist

Editoren/Fraise/SyntaxDefinitions.plist

### Hinweis

Diese Konfiguration wurde für den **Fraise 3.7.3** erstellt.

### Installation

- Gehen Sie in den Finder, wählen Sie Programme und öffnen Sie die "Fraise.app" mit einem Rechtsklick "Paketinhalt zeigen".
- Gehen Sie zu "Contents/Resources/". Öffnen Sie einen zweiten Finder und kopieren Sie die Konfigurationsdatei "SyntaxDefinitions.plist" in die Zwischenablage. Gehen Sie zurück zum 1. Finder ("Contents/Resources/") und fügen Sie die Zwischenablage in diesen Ordner ein und überschreiben Sie die enthaltene Konfigurationsdatei.
- Gehen Sie zu "Contents/Resources/Syntax Definitions/". Gehen Sie zum zweiten Finder und kopieren Sie die Konfigurationsdatei "xmlconverter.plist" in die Zwischenablage. Gehen Sie zurück zum 1. Finder ("Contents/Resources/Syntax Definitions/") und fügen Sie die Zwischenablage in diesen Ordner ein.
- Schließen Sie beide Finder.

### Syntaxhervorhebungen

Alle Skript-Dateien mit der Dateierdung ".xb" werden den Highlighter-Einstellungen „XML Converter“ automatisch zugeordnet.

### Siehe

[Fraise]



## NotePad++

---

### Kompatibilität

Windows

### Konfigurationsdateien

```
Editoren\NotePadPP\xml_converter.xml
```

### Hinweis

Diese Konfiguration wurde für **NotePad++ Version 7.6.6** (deutsche Version) erstellt.

### Installation

- Starten Sie NotePad++, wählen Sie das Menü "Sprachen" – "Eigene Sprache definieren ..." oder drücken Sie den Button "Eigene Sprache definieren...".
- Es erscheint der Dialog "Benutzerdefinierte Sprache". Drücken Sie hier den Button „Importieren“ und wählen Sie die Konfigurationsdatei „xml\_converter.xml“ aus.
- Starten Sie NotePad++ erneut.

### Siehe

[NotePad++]



## PSPad

---

### Kompatibilität

Windows

### Hinweis

Diese Anleitung ist nur für PSPad Versionen, die älter als Version 4.6.0 sind, vorgesehen. Ab Version 4.6.0. wird der XML-Converter direkt unterstützt. Siehe auch Kapitel PSPad Editor Version 5.0.3 (377).

Sollte der XML Converter deutlich neuer sein als die verwendete PSPad-Version, soll wird eine Installation (Aktualisierung) der Konfigurationsdateien empfohlen.

### Konfigurationsdateien

```
Editoren\PsPad\XB.INI
```

```
Editoren\PsPad\XB.DEF
```

### Hinweis

Diese Konfiguration wurde für den **PSPad Editor Version 4.6.2 (2750)**, deutsche Version, erstellt.

### Installation

- Kopieren Sie die Syntax-Konfigurationsdatei (.INI) in den Syntaxordner des PSPad. Dieser befindet sich im Programmverzeichnis (in Windows x64 "C:\Program Files (x86)" andernfalls "C:\Programme") im Unterordner ("PSPad editor\Syntax").
- Kopieren Sie die Kontext-Konfigurationsdatei (.DEF) in den Kontextordner des PSPad. Dieser befindet sich im Programmverzeichnis (in Windows x64 "C:\Program Files (x86)" andernfalls "C:\Programme") im Unterordner ("PSPad editor\Context").
- Starten Sie den PSPad Editor (als Administrator).
- Gehen Sie in das Menü "Einstellungen / Highlighter einstellen..." und wählen Sie in der linken Spalte einen der 4 letzten leeren Einträge "<not assigned>" aus (Anklicken). Danach wählen Sie im rechten Fenster (Karteireiter: "Spezifikation") in der Liste "Benutzer-Highlighter" den Eintrag "XB" aus (Anklicken). In der linken Spalte wird jetzt automatisch der Eintrag "<not assigned>" auf "XB" geändert. Aktivieren Sie diesen Eintrag durch Anwählen der Checkbox dieses Eintrages (Klick in die Checkbox). Bestätigen Sie diese Konfiguration mit dem Button "Anwenden" und danach mit "OK".
- Die Programmzuordnung zu den Dateitypen "xb" kann über das Menü "Einstellungen / Programm einstellen..." erfolgen. Wählen Sie



in der linken Spalte den Eintrag "Dateierweiterung zuordnen". Geben Sie unten im Eingabefeld "Erweiterung" die Endung ".xb" ein und bestätigen Sie mit "Hinzufügen". Nach erfolgter Eingabe können Sie mit dem Button "Alle Registrieren" den PSPad als Standardeditor für diese Dateitypen in Windows registrieren.

- Beenden Sie das Programm und starten Sie es erneut. Fertig.

### Syntaxhervorhebungen

Alle Skript-Dateien mit der Dateierweiterung ".xb" werden den Highlighter-Einstellungen „XB“ automatisch zugeordnet.

### Debuggen

Verwenden Sie die Compiler-Einstellungen des Highlighters oder eines Projektes zum Debuggen von Konvertierungsprozessen. Stellen Sie den „Log-Parser“ dazu wie folgt ein:

```
in Datei '%F' Zeile=%L, Zeichen=%C
```

Als Compiler tragen Sie zum Beispiel die Batchdatei Ihres Projektes ein:

```
C:\Programme\XML Converter\Projekt\Test.bat
```

und das Startverzeichnis:

```
C:\Programme\XML Converter\Projekt
```

und die Logdatei, die in der Batchdatei als Ausgabe definiert wurde:

```
C:\Programme\XML Converter\Projekt\Test.log
```

Wenn Sie die Skriptdatei:

```
C:\Programme\XML Converter\Projekt\xmlhtml.xb
```

mit dem Editor bearbeiten, können Sie mit „Strg+F9“ die Konvertierung starten und erhalten die Logausgabe im Logfenster des Editors angezeigt. Mit einem Klick auf eine Fehlermeldung, die auf eine Dateiposition verweist, wird die angegebene Datei geöffnet und die Position markiert. Wenn Sie ein Projekt definiert haben, wird für alle Dateien des Projektes mit „Strg+F9“ die Konvertierung gestartet und das Logausgabe ausgewertet.

### Siehe

[PSPad]



## TextPad

---

### Kompatibilität

Windows

### Konfigurationsdateien

Editoren\TextPad\XB.syn

### Hinweis

Diese Konfiguration wurde für den **Version 8.0.2** (deutsche Version) erstellt.

### Installation

- **32 Bit Programmversion**  
Kopieren Sie die Konfigurationsdatei (`XB.syn`) in den Systemordner des Programms TextPad. Dieser befindet sich im Programmverzeichnis (in Windows x64 "C:\Program Files (x86)" andernfalls "C:\Programme") im Unterordner ("TextPad 8\System").
- **64 Bit Programmversion**  
Kopieren Sie die Konfigurationsdatei (`XB.syn`) in den Systemordner des Programms TextPad. Dieser befindet sich im Programmverzeichnis ("C:\Program Files") im Unterordner ("TextPad 8\System").
- Starten Sie TextPad.
- Gehen Sie in das Menü "Konfiguration / Neue Dokumentklasse..." und tragen Sie hier "XB". Gehen Sie auf "Weiter" und tragen Sie den Dateityp (Elemente) wie folgt ein "\* .xb. Gehen Sie auf "Weiter" und aktivieren Sie "Syntax-Hervorhebungen einschalten" und wählen Sie in der Auswahlliste "Syntaxdefinitionsdatei" den Eintrag "XB.syn" aus. Folgen Sie den weiteren Anweisungen.
- Beenden Sie das Programm und starten Sie es erneut.

### Debuggen

Verwenden Sie in den Compiler-Einstellungen folgenden Eintrag für „Regulärer Ausdruck zur Sprungziel in Werkzeug-Ausgabe“:

```
^.+'\([^']+\)' Zeile=\([0-9]+\), Zeichen=\([0-9]+\)
```

und tragen Sie die Parameter „Register“ wie folgt ein:

```
Datei:1 Zeilen:2 Spalten:3
```

### Siehe

[TextPad]



## TextWrangler

---

### Kompatibilität

macOS

### Konfigurationsdateien

```
Editoren\TextWrangler\xml_converter.plist
```

### Hinweis

Diese Konfiguration wurde für den **TextWrangler 4.5.12** erstellt.

### Installation

- Kopieren Sie die Konfigurationsdatei (`xml_converter.plist`) in den Ordner  
`~/Library/Application Support/TextWrangler/Language Modules/`.  
Kopieren Sie dazu die Konfigurationsdatei in die Zwischenablage, wählen Sie im Finder "Gehe zu", halten Sie die "Alt"-Taste gedrückt, so dass in der Liste "Library" erscheint und wählen Sie "Library" aus. Jetzt wählen Sie  
`"Application Support/TextWrangler/Language Modules/"`.  
Fügen Sie den Inhalt der Zwischenablage hier ein.

### Hinweis

Der Benutzerordner in macOS wird mit "~" angegeben. Sie können auch "user/name" (für "name" bitte den Benutzerordnernamen angeben) verwenden.

### Syntaxhervorhebungen

Alle Skript-Dateien mit der Dateierdung ".xb" werden den Highlighter-Einstellungen „XML Converter“ automatisch zugeordnet.

### Siehe

[TextWrangler]



## Tincta

---

### Kompatibilität

macOS

### Konfigurationsdateien

Editoren/Tincta/XML Converter.plist

Editoren/Tincta/FileExtensions.plist

### Hinweis

Diese Konfiguration wurde für den **Tincta 3.1** erstellt.

### Installation

- Gehen Sie in den Finder, wählen Sie Programme und öffnen Sie die "Tincta.app" mit einem Rechtsklick "Paketinhalt zeigen".
- Gehen Sie zu "Contents/Resources/". Öffnen Sie einen zweiten Finder und kopieren Sie die Konfigurationsdatei "FileExtensions.plist" in die Zwischenablage. Gehen Sie zurück zum 1. Finder ("Contents/Resources/") und fügen Sie die Zwischenablage in diesen Ordner ein und überschreiben Sie die enthaltene Konfigurationsdatei.
- Gehen Sie zu "Contents/Resources/Syntax Definitions/". Gehen Sie zum zweiten Finder und kopieren Sie die Konfigurationsdatei "XML Converter.plist" in die Zwischenablage. Gehen Sie zurück zum 1. Finder ("Contents/Resources/Syntax Definitions/") und fügen Sie die Zwischenablage in diesen Ordner ein.
- Schließen Sie beide Finder.

### Syntaxhervorhebungen

Alle Skript-Dateien mit der Dateierdung ".xb" werden den Highlighter-Einstellungen „XML Converter“ automatisch zugeordnet.

### Siehe

[Tincta]



## UE Studio

---

### Kompatibilität

Windows

### Konfigurationsdateien

```
Editoren\UEStudio\xml_converter.uew
```

### Hinweis

Diese Konfiguration wurde für **UE Studio Version 19.00** (deutsche Version) erstellt.

### Installation (UEStudio)

- Kopieren Sie die Konfigurationsdatei („`xb.uew`“) in den Benutzerordner des Programms UEStudio. Dieser befindet sich im Benutzerverzeichnis ("C:\Dokumente und Einstellungen\{*Benutzername*}\Anwendungsdaten\IDMComp\UEStudio\wordfiles").
- Starten Sie UEStudio.

### Syntaxhervorhebungen

Alle Skript-Dateien mit der Dateiendung ".`xb`" werden der Syntaxhervorhebung „XB“ automatisch zugeordnet.

In der Funktionsleiste werden automatisch die Namen der Elemente und Entitäten aufgelistet.

### Siehe

UEStudio



## UltraEdit

---

### Kompatibilität

Linux, macOS, Windows

### Konfigurationsdateien

Editoren\UltraEdit\xb.uew

### Hinweis

Diese Konfiguration wurde für den **UltraEdit Version 26.00** (deutsche Version) erstellt.

### Installation (Linux)

- Kopieren Sie die Konfigurationsdatei (`xb.uew`) in den Ordner `~/ .idm/uex/wordfiles/`. (Benutzen Sie hierfür ein Dateiverwaltungsprogramm wie z.B. Kurader.)
- Starten Sie UltraEdit.

### Hinweis

Der Benutzerordner in Linux wird mit "~" angegeben. Sie können auch "home/name" (für "name" bitte den Benutzerordnernamen angeben) verwenden.

### Installation (macOS)

- Kopieren Sie die Konfigurationsdatei (`xb.uew`) in den Ordner `~/Library/Application Support/UltraEdit/wordfiles/`. Kopieren Sie dazu die Konfigurationsdatei in die Zwischenablage, wählen Sie im Finder "Gehe zu", halten Sie die "Alt"-Taste gedrückt, so dass in der Liste "Library" erscheint und wählen Sie "Library" aus. Jetzt wählen Sie `Application Support/BBEdit/UltraEdit/wordfiles/`. Fügen Sie den Inhalt der Zwischenablage hier ein.
- Starten Sie UltraEdit.

### Installation (Windows)

- Kopieren Sie die Konfigurationsdatei („`xb.uew`“) in den Benutzerordner des Programms UltraEdit. Dieser befindet sich im Benutzerverzeichnis (`C:\Dokumente und Einstellungen\{Benutzername}\Anwendungsdaten\IDMComp\UltraEdit\wordfiles`).
- Starten Sie UltraEdit.

**Syntaxhervorhebungen**

Alle Skript-Dateien mit der Dateierdung ".xb" werden der Syntaxhervorhebung „XB“ automatisch zugeordnet.

In der Funktionsleiste werden automatisch die Namen der Elemente und Entitäten aufgelistet.

**Siehe**

[UltraEdit]



# Systemanforderungen

## Betriebssysteme

- **Linux (ubuntu 16.04 LTS)** x32 und x64
- **macOS Catalina 10.15** x64
- macOS Mojave 10.14 x64
- macOS High Sierra 10.13 x64
- macOS Sierra 10.12 x64
- OS X 10.11 (El Capitan) x64
- OS X 10.10 (Yosemite) x64
- OS X 10.9 (Mavericks) x64
- OS X 10.8 (Mountain Lion) x64
- Mac OS X 10.7 (Lion) x64
- **Windows 10** x32 und x64
- Windows 8 x32 und x64
- Windows 7 x32 und x64
- Windows XP (Version 5.2) x64
- Windows XP (Version 5.1) x32

## Anforderungen an die Daten

Es werden beliebige XML-/SGML-Instanzen konvertiert, wobei folgende Einschränkungen bzw. Einstellungen zu beachten sind:

Es dürfen als Attributwerte nur *Zeichenketten* verwendet werden!

Die Instanz muss als vollständiges Dokument in einer Datei abgebildet sein – Parameterentitäten bzw. Texteneinschübe aus externen Dateien sind unzulässig.

Die Dokument-Typ-Deklaration (DTD) wird nicht ausgewertet. Es wird vorausgesetzt, dass die zu konvertierenden XML-/SGML-Instanzen entsprechend ihrer DTD valide sind.

Die DTD darf nicht im Dokument eingebettet sein.

Die Dateigröße darf 2.147.483.647 Zeichen (2 GB) nicht übersteigen. Bei 32-Bit-Systemen wird die Dateigröße zusätzlich durch den verfügbaren Hauptspeicher eingeschränkt. Die Dateigröße darf 1/8 bis 1/16 des verfügbaren freien Hauptspeichers betragen.

Die Größe der *Bezeichner* (*Attributname*, *Elementname* und *Entitätenname*) ist auf maximal 255 Zeichen begrenzt.

Die maximale Größe einer *Zeichenkette* (*Attributwerte*) ist auf 2.097.152 Zeichen (2 MB) begrenzt.

Ausführungsanweisungen dürfen in den Daten enthalten sein. Ausführungsanweisungen werden aber ignoriert. Die maximale Größe der Zeichendaten innerhalb einer Ausführungsanweisung ist auf 2.097.152 Zeichen (2 MB) begrenzt.

### XML

Alle Zeichen eines Dokumentes müssen entsprechend

- **ASCII**,
- **UTF-8**,
- **UTF-16** oder
- **UTF-32**

kodiert sein.

Wenn die Kodierung **ASCII** verwendet wird, dürfen nur Zeichen im unteren Zeichenbereich benutzt werden. (Zeichen kleiner 127) verwendet werden.

Andere Zeichenkodierungen dürfen nicht verwendet werden!

Folgende Deklarationen werden angenommen:

- **NAMECASE GENERAL=NO ENTITY=NO**

### SGML

Alle Zeichen eines Dokumentes müssen entsprechend ASCII (auch US-ASCII bzw. ANSI) kodiert sein. Folgende Zeichen sind zulässig:

9	(Tabulator)
10	(CR)
13	(LF)
32	(Leerzeichen)
33-126	!"#\$%&'()*+,-./0123456789:;<=>?@ ABCDEFGHIJKLMNPOQRSTUVWXYZ[\]^_` abcdefghijklmnopqrstuvwxyz{ }~

Die DTD wird **nicht** ausgewertet.

Die SGML-Deklaration wird **nicht** ausgewertet.

Folgende Einstellungen der SGML-Deklaration werden vorausgesetzt:

- **SHORTREF = NONE**
- **SHORTTAG = NO**
- **USEMAP =NO**
- **OMITTAG = NO/(YES)**
- **DATATAG = NO**
- **SUBDOC = NO**
- **CONCUR = NO**



- RANK = NO
- NAMELEN = 255
- LITLEN = 2097152 (2 MB)
- PILEN = 2097152 (2 MB)
- NAMECASE GENERAL=**YES** ENTITY=**NO**
- FUNCTION
  - RE** 13
  - RS** 10
  - TAB** 9
  - SPACE** 32

#### Deklarationen

- #DEFAULT nicht zulässig

#### DTD

- Elementinhalte dürfen in der DTD nicht als CDATA oder RCDATA deklariert werden. Alle Elementinhalte werden als #PCDATA gewertet, d.h., die Inhalte werden geparkt.
- Der Inhaltstyp "any" sollte vermieden werden.

Innerhalb des Dokumentes sind nicht erlaubt:

- Einbettung der DTD
- Einbettung der DEC (SGML-Deklaration)
- Elementdefinitionen <!ELEMENT ...>
- Attributdefinitionen <!ATTLIST ...>
- Deklaration von Notationen <!NOTATION ...>
- Entitätendefinitionen <!ENTITY ...> mit der Ausnahme der ENTITY N-DATA Deklaration vom Typ SYSTEM für externe Grafiken.
- Zeilenumbrüche und Tabulatoren werden innerhalb von #PCDATA-Bereichen als Inhalte gewertet. Außerhalb der #PCDATA-Bereiche werden diese ignoriert und nicht ausgegeben.
- Die Element- und Attributnamen können in beliebiger Groß/Kleinschreibung angegeben werden.

#### HTML

Es wird die HTML5-Syntax in purer Form unterstützt.

- XML-Erweiterungen sind nicht zulässig
- Es werden ausschließlich die Kodierungen, die über BOM im HTML-Dokument kodiert sind, unterstützt. Es wird empfohlen, generell UTF-8 zu verwenden.
- <Style> und <Script> sind zulässig, werden aber nicht ausgewertet oder ausgeführt.



Bitte beachten Sie auch die folgenden Hinweise zu wohlgeformten Daten (SGML und XML):

## Wohlgeformtes SGML

---

SGML Daten zu verarbeiten, erscheint auf den ersten Blick einfach. Aber nur auf den ersten Blick. So erlaubt die Komplexität von SGML Variationen der Daten, die für den ungeübten Anwender fälschlicherweise als Fehler erkannt werden können, aber "durch den Parser betrachtet" völlig korrekt sind. Zum Beispiel kann die Entität `&szlig;` wie folgt für den Parser korrekt eingesetzt werden: "Ich wei&szlig; es" aber auch "Ich wei&szlig es" (ohne abschließendes Semikolon nach `szlig`). Die Darstellung ohne abschließendes Semikolon wurde oft als Fehler betrachtet und in den Daten korrigiert, obwohl gemäß Deklaration diese Variation zulässig waren.

Mit diesem Werkzeug wurden im Jahr 1999 Anforderungen an die Daten gestellt, die eine lesbarere, einheitlichere Form der Daten ergeben sollte. Ein kleines Regelwerk schuf einheitliche Daten und auch die Grundlage für eine einfache Bearbeitung und Verwertung der Daten.

Heute kennen wir solche Regeln unter dem Begriff "Wohlgeformtheit". Ein großer Teil dieser Regeln entspricht den Regeln für wohlgeformte XML Dokumente, wie wir sie heute kennen:

- Die Dokument-Typ-Deklaration (DTD) wird nicht ausgewertet. Es wird vorausgesetzt, dass die zu konvertierenden SGML-Instanzen entsprechend ihrer DTD valide sind. Als Grundlage dafür müssen die Daten einheitlich geformt sein. Da für das Prüfen (Parsen) von SGML-Daten das Erkennen von leeren Elementen problematisch ist, wurde dafür ein Kennzeichen in das Skript implementiert. Damit ist es möglich, SGML auf Wohlgeformtheit zu prüfen. Das Problem der leeren Elemente wurde bereits 1997 erkannt und durch Web SGML (ISO 8879 TC2, Annex K Web SGML Adaptations, siehe [Web SGML]) gelöst, hat aber in SGML-Anwendungen außerhalb des Webs kaum Einzug gehalten
- Die Dokument-Typ-Deklaration (DTD) darf nicht im Dokument eingebettet sein. Um sauber Daten und Definition zu trennen, wurde mit dem Converter festgelegt, dass die DTD separat von den Daten zu halten ist. Die Daten sollten über den DOCTYPE der DTD zugeordnet werden. Ein Dokument besteht aus dem Prolog und der Dokumentinstanz.
- Alle Entitäten müssen mit `" ; "` abgeschlossen werden. SGML erlaubt das Abschließen einer Entität auch mit einem Whitespace-Zeichen, wie z.B. `"&sect 11"` statt `"&sect; 11"`. Das `" ; "` kann gemäß SGML in diesem Fall entfallen, da das Whitespace-Zeichen `" "` ein Entitäten-End-Signal aussendet und die Entität damit beendet wird. Für ein verbessertes Lesen und vereinfachtes Nacharbeiten der Daten ist das konsequente Abschließen aller Entitäten aber sinnvoll.
- Das `"&"`-Zeichen ist als Textzeichen nicht erlaubt und muss als Entität `"&amp"`, `"&#x26;"` oder `"&#38;"` ausgezeichnet werden.
- Das `"<"`-Zeichen ist als Textzeichen nicht erlaubt und muss als Entität `"&lt;"`, `"&#x3C;"` oder `"&#60;"` ausgezeichnet werden.

- Das ">"-Zeichen ist als Textzeichen nicht erlaubt und muss als Entität "&gt;", "&#x3E;" oder "&#62;" ausgezeichnet werden. Um Sicherzustellen, dass kein "verlorener Tag" in den Daten enthalten ist, wurden die Tag-Beginn("<")- und Tag-Ende(">")- Kennzeichen als Textdaten generell verboten. So sind Probleme der Form "<eintrag" oder "eintrag>" schnell zu finden.
- Attribute müssen stets vollständig angegeben werden, also in der Form `Attributname = Attributwert`. SGML erlaubt mit der Option SHORTTAG auch das Auslassen des Attributnamens, wenn dieser durch die DTD eindeutig zu erkennen ist.
- Die Attributwerte müssen stets als Zeichenketten angegeben werden. SGML erlaubt auch die direkte Angabe von Attributwerten ohne Zeichenkettenbegrenzer.
- Alle Tags müssen mit ">" beendet werden. In SGML ist es mit der Option SHORTTAG auch möglich, Tags zu verkürzen. Ein Tag wird demnach auch dadurch geschlossen, dass ein neuer Tag begonnen wird.
- Leere Tags sind nicht erlaubt. In SGML ist es mit der Option SHORTTAG möglich, ein Element mit einem leeren Endtag zu beenden "</>" anstatt den passenden Endtag zu verwenden. Ebenfalls kann ein Element mit einem leeren Starttag "<>" begonnen werden, wenn es durch die DTD eindeutig erkannt werden kann. Zur eindeutigen Erkennung und zur verbesserten Lesbarkeit ist die Angabe von Starttags und Endtags zwingend erforderlich, mit Ausnahme des leeren Elementes, welches nur durch den Starttag repräsentiert wird.
- Endtags für leere Elemente sind nicht erlaubt. In SGML können leere Elemente optional mit einem Endtag geschlossen werden. Um eine eindeutige Schreibweise zu erzielen wurde diese Regel eingeführt. *Diese Regel ist etwas restriktiver, als die vergleichbare XML Regel. In XML kann ein leeres Element entweder durch einen Starttag gefolgt von einem Endtag gekennzeichnet werden oder durch ein Empty Element Tag "<.../>".*

- Alle Zeichen eines Dokumentes müssen entsprechend ASCII (auch US-ASCII bzw. ANSI) kodiert sein. Nur folgende Zeichen sind zulässig:

```

9      (Tabulator)
10     (CR)
13     (LF)
32     (Leerzeichen)
33-126 !"#%&'()*+,-./0123456789:;<=>?@
      ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`
      abcdefghijklmnopqrstuvwxyz{|}~
    
```

Diese Datenkodierung gewährleistet eine 100%ige Sicherstellung der korrekten Anzeige in allen Werkzeugen, SGML-Editoren und beliebigen Texteditoren.

Dieser Teilbereich des Zeichensatzes ist kompatibel mit:



ANSI, ASCII, US-ASCII,  
ISO-8859-1 bis ISO 8859-15,  
UTF-8.

Auch wenn später in XML der Zeichenbereich komplett geöffnet wurde (UTF-8, UTF-16, usw.), wird durch die vorgenannte Regel der eingeschränkte Zeichenbereich weiterhin unterstützt. Eine Lesbarkeit der Daten ist in der Zukunft garantiert.

## Wohlgeformtes XML

Die Spezifikationen entsprechend dem Kapitel Wohlgeformtes SGML gelten natürlich auch für XML. Im Jahr 2000 wurden folgende Anpassungen vorgenommen, um zusätzlich zu SGML-Daten auch XML-Daten verarbeiten zu können:

- Der Namensraum für Bezeichner wurde um die Zeichen "-", "\_ " und ":" erweitert.
- Endtags für leere Elemente sind nicht erlaubt. In XML kann ein leeres Element entweder durch einen Starttag gefolgt von einem Endtag gekennzeichnet werden oder durch ein Empty Element Tag "<.../>". Für den Converter dürfen nur Daten, deren leere Elemente mit dem Empty-Element-Tag gekennzeichnet sind, verwendet werden. Diese Regel ist etwas restriktiver, als die vergleichbare XML Regel.

Alternativ ist es durchaus möglich, leere Elemente durch einen Starttag und einen darauf folgenden Endtag zu kennzeichnen. Diese Syntaxform entspricht der Syntax für Elemente, wobei der Inhalt lediglich ausgelassen wird. Es darf aber nur eine Syntaxform im Dokument verwendet werden! Mischformen sind nicht zulässig!

Abweichend von den XML Regeln sind folgende zusätzliche Einschränkungen für die Daten zu beachten:

- Das ">"-Zeichen ist als Textzeichen nicht erlaubt und muss als Entität "&gt;";", "&#x3E;";" oder "&#62;";" ausgezeichnet werden. Um sicherzustellen, dass kein "verlorener Tag" in den Daten enthalten ist, wurden die Tag-Beginn("<")- und Tag-Ende(">")- Kennzeichen als Textdaten generell verboten. So sind Probleme der Form "<eintrag" oder "eintrag>" schnell zu finden.
- Alle Zeichen eines Dokumentes müssen entsprechend ASCII (auch US-ASCII bzw. ANSI) kodiert sein. Nur folgende Zeichen sind zulässig:
 

9	(Tabulator)
10	(CR)
13	(LF)
32	(Leerzeichen)
33-126	!"#\$%&'()*+,-./0123456789:;<=>?@ ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_` abcdefghijklmnopqrstuvwxyz{ }~

Diese Datenkodierung gewährleistet eine 100%ige Sicherstellung der korrekten Anzeige in allen Werkzeugen, SGML-Editoren und beliebigen Texteditoren.

Dieser Teilbereich des Zeichensatzes ist kompatibel mit:

ANSI, ASCII, US-ASCII,  
 ISO-8859-1 bis ISO 8859-15 und  
 UTF-8.

Auch wenn in XML der Zeichenbereich komplett geöffnet wurde (UTF-8, UTF-16, usw.), wird durch die vorgenannte Regel der eingeschränkte Zeichenbereich weiterhin unterstützt. Eine Lesbarkeit der Daten ist in der Zukunft garantiert.

Seit 2009:

- Der Zeichenbereich der Daten kann für XML-Anwendungen wahlweise "ASCII" oder "UTF-8" sein. Die Vorgabe ist normgerecht UTF-8.

Einschränkung:

Die Regeln des eingeschränkten Zeichenbereiches gelten aber weiterhin für alle Strukturelemente von XML: Element-, Attribut- und Entitätenbezeichner. UTF-8 kann nur im #PCDATA-Bereich bzw. innerhalb von Attributwerten verwendet werden. UTF-8 wird hierbei nicht ausgewertet.

Seit 2011:

- Der Zeichenbereich der Daten kann für XML-Anwendungen zusätzlich auch UTF16, UTF-16BE, UTF-16LE, UTF-32, UTF-32BE oder UTF-32LE sein..

Einschränkung:

Es werden nur Zeichen der Unicode Plane 0 (Basic Multilingual Plane, BMP, x0 bis xFFFF) unterstützt.

Seit 2018:

- Der Zeichenbereich der Daten wird vollständig gemäß Unicode unterstützt x0 bis x10FFFF.

Seit 2019:

- Die Zeichenbereiche für XML-Bezeichner (Elementnamen und Attributnamen) wurden auf die Bereiche xC0-xD6 und xD8-xF6 und xF8-xFF erweitert, womit z.B. deutsche Umlaute hier zulässig sind.





# Entwicklungsgeschichte

<i>Version</i>	<i>Datum - Beschreibung</i>
4.0.10	17.9.2017 Korrekturen: <ul style="list-style-type: none"><li>• Kleinere Fehlerbeseitigungen in den Logausgaben.</li></ul>
4.0.9	1.9.2017 Korrekturen: <ul style="list-style-type: none"><li>• Windows-Version: Umstellung des Compilers auf Visual Studio 2017.</li><li>• Korrekturen im Handbuch.</li></ul> Updates <ul style="list-style-type: none"><li>• NotePad++ 7.3.3 vom 8.3.2017</li><li>• TextPad Text Editor Version 8.1.2 vom 10.3.2017</li></ul>
4.0.8	2.4.2017 Korrekturen: <ul style="list-style-type: none"><li>• SGML-Parser: Leerräume (Whitespace) zwischen Attributen sind jetzt zwingend erforderlich. Hinweis: In SGML ist der Leerraum zwischen Attributen optional definiert. Beispiel: <code>&lt;elem attr1="wert1"attr2="wert2"&gt;</code> ist in SGML nach Vorschrift gültig, wird aber in der Praxis sinnvollerweise nicht genutzt. Der XML-Converter erwartet jetzt stets einen Leerraum zwischen Attributen. Beispiel: <code>elem attr1="wert1" attr2="wert2"&gt;</code>.</li></ul>
4.0.7	21.3.2017 Korrekturen: <ul style="list-style-type: none"><li>• Korrektur der FILE-Funktion. Das Ersetzen der Dateierweiterung wurde für den Fall von relativen Rückwärtspfaden korrigiert.</li></ul> Updates: <ul style="list-style-type: none"><li>• PSPad 4.6.2 (2750) vom 1.3.2017</li></ul>

4.0.6

30.10.2016

Korrekturen:

- Korrektur der numerischen Attributwertsvergleichfunktion. Korrektur und Erweiterung der Größenbeschränkung. Korrektur der Einheitenbewertung. Die Einheiten werden jetzt vollständig ausgewertet, eventuell nach der Einheit stehende Zeichen werden jetzt mit berücksichtigt. Der numerische Vergleich wird erst dann durchgeführt, wenn die Einheiten beider Vergleichswerte gleich sind unabhängig der Groß-/Kleinschreibung.
- Der numerische Vergleich ist jetzt direkt mit Werten unterschiedlicher Einheiten möglich. Die Einheiten pt, em (wird ersatzweise mit 10pt berechnet), mm, cm und in werden erkannt. Wenn beide Vergleichswerte jeweils eine der angegebenen Einheiten enthalten, werden die Vergleichswerte intern in pt umgerechnet und numerisch verglichen.

Updates:

- BBEdit Version 11.6.2 (397016) vom 15.9.2016

## 4.0.5 12.10.2016

## Neues:

- No-Spy-Code für die Windowsversion. In der Windowsversion wurde die von Microsoft Visual Studio 2015 automatisch implementierte sogenannte Telemetriefunktion deaktiviert. Das Einbinden von Microsoft Telemetrieprogrammcode in den Konverter kann möglicherweise aber nicht verhindert werden. Diese Telemetriefunktion ist in Microsoft Visual Studio 2015 bis einschließlich der Version Microsoft Visual Studio Update 2 enthalten. Siehe Artikel: <http://www.heise.de/newsticker/meldung/Visual-Studio-2015-stopft-ungefragt-Tracing-Code-in-C-Programme-3235676.html> und [https://www.reddit.com/r/cpp/comments/4ibauu/visual\\_studio\\_adding\\_telemetry\\_function\\_calls\\_to\\_d30dmvu](https://www.reddit.com/r/cpp/comments/4ibauu/visual_studio_adding_telemetry_function_calls_to_d30dmvu).
- Die Windowsversion wurde mit der aktualisierten Version von Microsoft Visual Studio Update 3 (KB3165756) vom 14.9.2016 neu kompiliert. Die bisherige Telemetriefunktion ist (offenbar) von Microsoft entfernt worden.
- Detailliertere Versionsausgabe für das Entwicklungssystem der Windowsversion.
- Kompatibilität mit macOS Sierra Version 10.12 hergestellt.

## Updates:

- PSPad 4.6.1 (2730) vom 4.7.2016
- Notepad++ 6.9.2 vom 18.5.2016
- TextPad 8.0.2 vom 27.2.2016
- UltraEdit 23.10 vom 23.5.2016
- UEStudio 16.10 vom 23.5.2016
- TextWrangler 5.5.2 vom 10.9.2016
- BBEdit 11.5.2 vom 21.4.2016
- Tincta 3.1.1 vom 26.9.2016

## 4.0.4 8.5.2016

## Korrekturen:

- Reguläre Ausdrücke funktionieren jetzt auch unter Windows XP.

## 4.0.3 31.1.2016

## Korrekturen:

- Fehlerkorrektur bei komplexen regulären Ausdrücken.



4.0.2 25.11.2015

Korrekturen:

- Der Rekursionsschutz wurde verbessert.

4.0.1 22.11.2015

Korrekturen:

- Fehlerkorrektur im Befehl TOKEN.
- Die minimale Konsolenausgabe (Schalter -q) wurde so geändert, dass bei Erfolg keine Konsolenausgabe erfolgt.

4.0.0

17.10.2015

Neu:

- Der Vergleich eines Attributwertes mit einer Zeichenkette wurde um Vergleichsoperatoren für reguläre Ausdrücke erweitert: `=~` Attributwert wird gegen den regulären Ausdruck geprüft, Groß- und Kleinschreibung wird hierbei ignoriert, `==~` Attributwert wird gegen den regulären Ausdruck unter Beachtung der Groß- und Kleinschreibung geprüft.
- Der Vergleich eines Attributwertes mit einer Zeichenkette wurde um neue Vergleichsoperatoren erweitert: `=^` Zeichenkette beginnt mit angegebenen Text, `=*` Zeichenkette enthält angegebenen Text, `=$` Zeichenkette endet mit angegebenen Text. Diese Operatoren berücksichtigen keine Groß-/Kleinschreibung. Wenn der Vergleich unter Beachtung der Groß-/Kleinschreibung erfolgen soll, müssen diese wie folgt angegeben werden: `==^`, `==*` und `==$`.
- Der Vergleich von Attributwerten wurde um die automatische Erkennung von Zahlenwerten und den numerischen Vergleich erweitert. So werden jetzt numerische Werte numerisch verglichen, das betrifft "`=`", "`>`", "`<`", "`<>`", "`>=`" und "`<=`". Die Werte werden numerisch interpretiert, die Werte "15", "15.0", "15,00" und "015" sind also gleichwertig. Hat der Vergleichswert im Skript eine Einheit (z.B. "15px"), wo wird der Vergleich nur dann numerisch durchgeführt, wenn das betreffende Attribut ebenfalls diese Einheit trägt (z.B. 155 px).
- Erhöhung der maximalen Größe für Attributwerte auf 2 MB, `LITLEN=2.097.152`
- Erhöhung der maximalen Größe für Ausführungsanweisungen auf 2 MB, `PILEN=2.097.152`
- Fit für Windows 10.

Korrekturen

- Bei numerischen Auswertungen von Attributwerten werden die Zahlenformate automatisch erkannt (deutsch mit Dezimalpunkt Komma und englisch Dezimalpunkt Punkt).
- Die `IMPLIED`-Anweisung erzeugt keine Fehlermeldung mehr, wenn das betreffende Attribut nicht gefunden und das Attribut auch nicht definiert wurde.
- Korrekturen im Handbuch.

Editor Konfigurationen:

- Aktualisierung für PSPad  
(Hinweis: ab PSPad Version 4.6.0 (2566) vom 29.3.2015)

- 3.0.1 23.6.2013  
Neu:
- Version für Linux.
  - 32 Bit- und 64 Bit-Version für Linux verfügbar.
  - Hinweis: Der Kompatibilitätsmodus ist nicht in der Version für Linux verfügbar.
- 3.0.0 7.4.2013  
Neu:
- Version für Mac OS X.
  - 32 Bit- und 64 Bit-Version für Mac OS X verfügbar.
  - Hinweis: Der Kompatibilitätsmodus ist nicht in der Mac OS X Version verfügbar.
  - Unterstützung für UNC-Pfade (\\Server\Pfad).
- Änderung der Kommandozeile:
- Die Kommandozeilenschalter werden jetzt generell mit - (Strich) statt / (Schrägstrich) eingeleitet.
  - Alle Kommandozeilenparameter müssen mit einem Leerzeichen voneinander abgetrennt werden. Ältere Batchprogramme müssen also wie folgt umgestellt werden, statt z.B. "/lg/o" muss "-lg -o" angegeben werden.
  - Der Aufruf der Hilfe erfolgt über den Parameter -h, statt zuvor mit "-?". Weitere Parameter werden ignoriert.
- 2.1.0 2.2.2013  
Erweiterungen
- Kodierung UTF-8 vollständig implementiert (lesen und schreiben).
  - Writer Kodierung ASCII (7 Bit, Zeichen größer ab 128 werden als Zeichenverweise &#x80; usw. ausgegeben).
  - Writer Kodierung BIN (8 Bit, Zeichen größer ab 256 werden als Zeichenverweise &#x100; usw. ausgegeben).
  - Konfiguration für PSPad 4.5.7
  - Konfiguration für Notepad++ 6.2.3
  - Konfiguration für TextPad 6.2.2
  - Konfiguration für UltraEdit 18.20
  - Konfiguration für CodePad2 2.0.4.0

- 2.0.3 9.4.2012  
Erweiterungen:
- CHR auch für Unicodezeichen (Plane 0).
  - Schalter -d jetzt auch in Kombination mit Schalter -q nutzbar.
  - Reader-Encoding für UTF-8, UTF-16 und UTF-32 und AUTO
  - Writer-Encoding UTF-16 und UTF-32.
- Korrekturen:
- Korrektur der EMPTY-Abfrage (Unterelemente werden jetzt korrekt als Inhalt gewertet).
- 2.0.2 27.11.2011  
Erweiterungen:
- 64 Bit Version
  - Konfiguration für Editor PSPad 4.5.6.
- Korrekturen:
- Prüfung auf korrekte ASCII-Kodierung wurde auf SGML oder auf XML mit Encoding ASCII eingeschränkt
- 2.0.1 25.9.2011  
Korrekturen:
- Probleme mit der SGML-Dekodierung der Unicodeversion 2.00.0 behoben.

- 2.0.0      18.9.2011
- Umstellung auf Unicode
- Unicode-Version 32 Bit (beta)
- Erweiterungen:
- Dekoder für UTF-8  
(8 Bit, eingeschränkt auf Basic Multilingual Plane)
  - Dekoder für UTF-16  
(16 Bit, eingeschränkt auf Basic Multilingual Plane, LE und BE)
  - Dekoder für UTF-32  
(32 Bit, eingeschränkt auf Basic Multilingual Plane, LE und BE)
  - Dekoder für UCS-2  
(16 Bit, LE und BE)
  - Dekoder für UCS-4  
(32 Bit, eingeschränkt auf Basic Multilingual Plane, LE und BE)
- Hinweis zur Betaversion:
- Die Eingabe-Dekodierung erfolgt stets automatisch. Es wird der BMP-Unicode-Bereich unterstützt (Plane 0). In künftigen Versionen wird dieses Verhalten per Skript steuerbar sein.
  - Die Ausgabe-Kodierung ist stets ASCII. Zeichen außerhalb des ASCII-Bereiches werden in der Beta-Version als Zeichenverweise (z.B. `&#x1AD;`) ausgegeben. In künftigen Versionen wird dieses Verhalten per Skript steuerbar sein.
- 1.29.6      26.5.2011
- Korrekturen:
- Darstellung der Umlaute in der Konsolenausgabe korrigiert.

- 1.29.5 11.5.2011  
Erweiterungen:
- Fehlermeldungen bei UTF8-Kodierungsfehlern verbessert.
  - Syntax der Kommandozeilenschalter für mehr Systemunabhängigkeit geändert (-c -lg). Die alte Syntax (/c/lg) ist weiterhin gültig. Achtung: Zwischen den Kommandozeilenparametern nach neuer Syntax sind jetzt Leerzeichen zwingend erforderlich.
  - Prüfmodus integriert. Dazu muss einfach die Angabe der Ausgabe weggelassen werden.
  - Konfigurationsdatei für TextPad 4.5.2, UltraEdit 17.00.0, NotePad++ 5.8.7, PsPad und CodePad 4.1 aktualisiert und Konfigurationsdatei für Codepad 2.0.2.0 neu hinzugefügt.
  - Operation "number" in TOKEN hinzugefügt, um reine Zahlenfolgen zu trennen bzw. Zahlenfolgen aus Texten zu extrahieren.
- Korrekturen:
- Stabilität verbessert (Probleme bei EMPTY-Abfrage in bestimmten Datensituationen beseitigt).
  - Probleme mit TOKEN bei der Aufspaltung von Zahlenfolgen mit Dezimalstellen behoben.
  - Probleme mit TOKEN bei Prozentberechnung und gleichzeitiger Formatangabe behoben.
  - Probleme mit hexadezimalen Zeichenreferenzen behoben.
- 1.29.4 12.1.2011  
Korrekturen:
- Fehlerkorrektur in FIND und COPY.
- 1.29.3 1.1.2011  
Erweiterungen:
- FIND, COPY, CUT und DEL mit neuem Suchbereich: CHILD.
  - FIND und COPY mit neuem Suchbereich: PARENT.
- 1.29.2 30.12.2010  
Lizenzbedingungen geändert. Freeware. Weitergabe möglich. Siehe Lizenz.txt.

- 1.29.1 12.12.2010  
Erweiterungen:
- FIND jetzt mit identischen Sucheinstellungen wie COPY, CUT und DEL.
  - Aktualisierung der Context-Profile für Editoren.  
Dokumentation zum Debuggen mit Unterstützung einiger Editoren.
- 1.28.9 26.9.2010  
Erweiterungen:
- Mit NAMECASE kann die Groß-/Kleinschreibung der Bezeichner (Elementnamen und Attributnamen) gesteuert werden. In SGML-Daten lassen sich somit Daten vergleichbar mit XML unter Beachtung der Groß-/Kleinschreibung prüfen. In XML kann abweichend von der Norm die Groß-/Kleinschreibung ignoriert werden.
  - Erweiterte Kontextabfrage BEFOREX zur Prüfung, ob das aktuelle Element ein Vorgängerelement zum angegebenen Element ist.
  - Erweiterte Kontextabfrage AFTERX zur Prüfung, ob das aktuelle Element ein Nachfolgeelement zum angegebenen Element ist.
  - VALUE, IMPLIED und TOKEN geben Zahlen in der Voreinstellung mit minimierten Dezimalstellen aus (Ganzzahlen ohne Dezimalpunkt, ansonsten nur die vorhandenen Dezimalstellen bis maximal 8 Dezimalstellen).
- Korrekturen:
- Die Berechnung mit VALUE mit negativem Offset ohne Formatangabe wurde korrigiert.
- 1.28.8 22.9.2010  
Erweiterungen:
- Befehl WARNINGS zur Steuerung des Verhaltens bei aufgetretenen Fehlern.
- Korrekturen:
- Meldung bei fehlerhafter SYNTAX-Angabe korrigiert.
- 1.28.7 30.08.2010  
Erweiterungen:
- Kommandozeilenschalter /q zur Aktivierung einer minimalen Konsolenausgabe (stiller Modus, quiet).

- 1.28.6 24.8.2010  
Erweiterungen:
- Der Attributvergleich zweier Attributwerte ist jetzt in allen Attributabfragen möglich. Dazu wird in einer Attributabfrage als 1. Parameter der Attributname des aktuellen Elementes angegeben und als 2. Parameter ein weiterer Attributname. Innerhalb von Kontextabfragen wird das 2. Attribut vom Kontextelement gelesen. So kann hiermit auch eine einfache REFID zu ID Abfrage realisiert werden.
- 1.28.5 16.1.2010  
Erweiterungen:
- Für SGML wurden die Funktionszeichenverweise `&#RE;`, `&#RS;`, `&#TAB;` und `&#SPACE;` implementiert. Die Definition im Skript kann über den Befehl ENTITY und den Funktionsnamen oder über die numerischen Werte erfolgen. Es darf allerdings nur eine Definition pro Zeichen im Skript definiert werden (z.B. `#TAB` oder `#9`).
  - kumulierte Fehlermeldungen bei Doppeldefinitionen von Elementen und Entitäten, Zeichenverweisen und Funktionszeichenverweisen im Skript.
- Korrekturen:
- Ein fehlender Vorgabewert eines Attributes erzeugt keine Fehlermeldung mehr.
  - Die Auswertung von Zeichenverweisen (dezimal und hexadezimal) wurde korrigiert.
- 1.28.4 6.9.2009  
Erweiterungen:
- XML: Markierte Bereiche bzw. CDATA-Bereiche implementiert,
  - SGML: Markierte Bereiche vom Typ CDATA, RCDATA, IGNORE, INCLUDE und TEMP vollständig implementiert.
- Korrekturen:
- SGML: Typprüfung der Attributwerte korrigiert.

- 1.28.3 23.8.2009  
Erweiterungen:
- Konfiguration für neue Version von Notepad++ 5.4.5.
  - Konfiguration für neue Version von PSPad 4.5.4.
  - Konfiguration für neue Version von TextPad 5.3.1.
  - Konfiguration für neue Version von UltraEdit 15.10.0.
- Korrekturen:
- Ausführung von COPY mit Parameter DOC korrigiert.
- 1.28.2 19.4.2009  
Erweiterungen:
- Bei der Bearbeitung werden keine temporären Dateien mehr erzeugt. Das Programm arbeitet jetzt vollständig im Arbeitsspeicher.
- Korrekturen:
- Probleme mit einstelligen hexadezimalen Zeichenreferenzen beseitigt.
- 1.28.1 29.3.2009  
Erweiterungen:
- Generelle Umstellung der internen Speicherverwaltung und der Speicherstrategien. Der Speicherbedarf ist gegenüber den Vorversionen erhöht, die Laufzeit des Skriptes wurde aber deutlich verbessert.
  - Erkennung von doppelten Definitionen von Elementen, Entitäten und Zeichenreferenzen.
  - Anzeige der verwendeten Syntax (SGML oder XML).
- Korrekturen:
- Innerhalb der Betriebsart "Konvertiermodus" (CONVERTENTITIES=ONLYDEFINED) wurde die Konvertierung von hexadezimalen Zeichenreferenzen korrigiert.
  - Probleme von COUNT im Suchbereich DOC beseitigt.

1.27.4

22.3.2009

Erweiterungen:

- In Verzweigungen (`IF-ELSEIF-ELSE-ENDIF`) sind jetzt auch leere Anweisungsblöcke erlaubt (während der Skriptprogrammierung nötig, um Anweisungsblöcke temporär auszukommentieren).
- Leerelemente in XML werden jetzt gemeinsam als Start- und Endtag geparkt und konvertiert. Dabei werden die Regeln für `BEG` und `END` nacheinander ausgeführt. So können mit dem gleichen Regelsatz die Elemente `<p>Text</p>`, `<p></p>` und `<p/>` konvertiert.

Korrekturen:

- Im Befehl `COPY` wurde die Ausführung der Elementregeln beim Ersetzungstyp `DROP` korrigiert.
- Die Fehlermeldung bei leeren Attributwerten wurde beseitigt.

1.27.3

15.3.2009

Erweiterungen:

- In `COPY` wurden die Ersetzungstypen `ALTFULL`, `ASCFULL` und `DROP` vom Kompatibilitätsmodus in die Standardsyntax übernommen.
- Das Basiselement `BASE` kann jetzt wahlweise als Elementname oder Zeichenkette angegeben werden.

Korrekturen:

- Die globale Ausgabe (mehrere XML-Dateien in eine Ausgabedatei konvertieren) wurde korrigiert.

1.27.2 7.3.2009

Erweiterungen:

- Skript kann als XML-Dokument erstellt werden. Als Basiselement dafür wird das Element `<script>` verwendet. Über die XML-Deklaration kann die Kodierung gesteuert werden, wobei "ASCII" oder "UTF-8" erlaubt sind.
- Parameter `/w` zum Aktivieren der Webfehlermeldungen direkt in die Ausgabedaten (HTML). Achtung: Dieser Parameter ist nicht im Kompatibilitätsmodus verfügbar.
- `LITLEN` (maximale Größe von Zeichenketten) wurde von 2048 auf 4096 (4 KByte) erweitert.
- `FILEN` (maximale Größe von Verarbeitungsanweisungen) wurde von 2048 auf 16384 (16 KByte) erweitert.
- Syntaxhervorhebung für TextPad.
- Syntaxhervorhebung für UltraEdit.
- Syntaxhervorhebung für CodePad.
- Syntaxhervorhebung für NotePad++.
- Handbuch erweitert.

Korrekturen:

- XML Zeichenbereich für Bezeichner um die Startzeichen ":" und "-" erweitert.
- Prüfung der Übereinstimmung der Kodierung von Skript und Daten und Ausgabe einer Fehlermeldung bei unterschiedlicher Kodierung.
- Prüfung des Basiselementes `BASE` verbessert.

1.27.1

8.2.2009

## Erweiterungen:

- Die Vorgabe für `CONVERTELEMENTS` ist jetzt `ONLYDEFINED`, im Kompatibilitätsmodus noch `ALL`.
- Die Vorgabe für `CONVERTENTITIES` ist jetzt `ONLYDEFINED`, im Kompatibilitätsmodus noch `ALL`.
- Die Konvertereinstellungen sind jetzt in beliebiger Reihenfolge erlaubt.
- In `FILE` wurde die Dateinamenserweiterung auf optional geschaltet.
- In `COPY` ist jetzt auch das eigene Element kopierbar. Dazu muss der Parameter für den Elementnamen einfach weggelassen werden und die Parameter mit einem Komma begonnen werden.
- Parameterklammern für `IN`, `INX`, `AFTER` und `BEFORE` für Kombinationen. Ohne Klammern wird das letzte Element als Ausgangsbasis für die nächste Bedingung genommen. Die Abfrage `IF AFTER eintrag BEFORE eintrag` prüft, ob sich das aktuelle Element hinter einem Element `eintrag` befindet und ob dieses Element `eintrag` vor einem Element `eintrag` befindet. Die Abfrage `IF AFTER(eintrag) BEFORE(eintrag)` prüft dagegen, ob sich das aktuelle Element hinter einem Element `eintrag` und auch vor einem Element `eintrag` befindet.
- Bei `CUT` und `DEL` werden die Start- und Endtags erhalten, wenn die in den Ersetzungstypen/Löschtypen ungleich `FULL` definiert sind.
- `COPY` und `CUT` erhalten als letzten Optionalen Parameter einen Trenntext, der zwischen die einzelnen Treffer als Text in die Ausgabe eingefügt wird (adäquat den alten `COPYALL`, `ASCALL`, `ALTALL`).
- Numerische Zeichenreferenzen (Dezimal oder hexadezimal) werden jetzt numerisch ausgewertet. In der `ENTITY`-Definition kann der numerische Wert dezimal `#123` oder hexadezimal `#xab` angegeben werden. Trifft der Parser auf eine Zeichenreferenz, z.B. `&#000123;`, so werden die `ENTITY`-Regeln abgearbeitet, deren numerischer Wert diesem entspricht (unabhängig der Darstellung `#123` oder `#00123` oder `#x7B` oder `#x007b`).
- Die XML-Kodierung wurde normgerecht auf UTF-8 eingestellt. Eine Kodierungsprüfung wurde für XML implementiert.
- Der `CHR`-Befehl schreibt beliebige Zeichen oder Bytes

- 1.26.4 18.1.2009  
Überarbeitung des Handbuchs.  
Erweiterungen:
- Mehrere `SYSTEM`-Definitionen im Skript erlaubt.
  - `SYSTEM`-Definition im Skript optional.
  - Mehrere `BASE`-Definitionen im Skript erlaubt.
  - `BASE`-Definition im Skript optional.
  - `COPY` und `FIND` jetzt auch mit Attributbedingung.
- Korrekturen
- Prüfung der Kontextbefehle und weitere Korrekturen der `NOT`-, `COPY`- und `FIND`-Befehle.
- 1.26.3 14.1.2009  
Korrekturen:
- Im Prolog wurde der `SYSTEM`-Identifikator direkt nach einem `PUBLIC`-Identifikator, abweichend von XML-Norm, wieder optional geschaltet (wie SGML).
  - Der XML-Parser wurde intern erweitert.
- 1.26.2 09.01.2009  
Korrekturen:
- Probleme mit `NOT`-Befehlen beseitigt.
- Skript:
- `NOTEEMPTY` hinzugefügt.
- 1.26.1 08.01.2009  
Probleme mit leeren erweiterten Vereinbarungen im `DOCTYPE` der Dokumente beseitigt.

- 1.25.1      31.1.2008 bis 18.12.2008
- bis
- 1.25.40
- Vorbereitung der nichtkommerziellen Version. Entfernung des Rechtemanagements. XML-Kompatibilität verbessert. Parser verbessert. Fehlermeldungen vereinheitlicht.
- Erweiterung: Konvertierung von Dokumenten mit nichtdefinierten Elementen und Entitäten jetzt auch im SGML-Modus verfügbar.
- Syntaxdateien für externe Editoren wurden vorbereitet.
- Generelle Handbuchüberarbeitung.
- Daten:
- NAMELEN wurde von 40 auf 255 erweitert.
  - LITLEN wurde von 1024 auf 2048 erweitert.
  - PILEN wurde von 255 auf 2048 erweitert.
  - Der gültige Zeichenbereich wurde um das Zeichen ~ (Tilde) erweitert.
- Korrekturen:
- VALUE- und IMPLIED-Abfragen jetzt ohne Fehlermeldung bei Attributen mit fehlendem Vorgabewert. Korrekte Vergleichswerte auch bei leeren Attributwerten bzw. Vorgabewerten.
  - Die Fehlermeldung bei Attributen in den Daten, die keine Entsprechung im Skript haben, wurde entfernt.
- Skript:
- Die Einstellung CONVERTELEMENTS erhielt den neuen Wert DROPUNDEFINED. So lassen sich alle im Skript undefinierten Elemente in der Ausgabe übergehen.
  - Die Einstellung CONVERTDATA wurde hinzugefügt. Damit lassen sich Leerbereiche (Whitespaces) unterdrücken.
  - Funktionsänderung der bestehenden Operatoren: "=" und "<>": (gleich und ungleich) unabhängig der Groß- und Kleinschreibung
  - Neue Vergleichsoperatoren "==" und "!=" eingeführt: (identisch und nicht identisch) abhängig der Groß- und Kleinschreibung.
  - Neuer Befehl IMPLIED für vererbte Attribute (als Abfrage und als Ausgabeanweisung).
  - Neuer Abfrage VALUE adäquat zu IMPLIED für Attribute (als Abfrage zusätzlich zur Ausgabeanweisung).
  - VALUE- und IMPLIED-Ausgabeanweisungen jetzt auch mit Elementangabe möglich.
  - Formatparameter für VALUE, IMPLIED und TOKEN

## XML Converter

### Produktname: „mediaTEXT XML Converter“

- 1.24 beta 16.1.2003  
Testversion mit optimierten Kontextabfragen.
- 1.23 16.1.2003
- Probleme bei der Prüfung der Attributdefinitionen aus Version 1.22 beseitigt.
  - Fehler bei Prüfung der Attributdefinitionen führen nicht mehr zum Abbruch, es werden lediglich Fehlermeldungen ausgegeben.
  - Probleme beim Start mit fehlerhaften Pfadbezeichnungen beseitigt.
- 1.22 28.11.2002
- Statusrückmeldung an Konsole über "errorlevel"  
0: Erfolg  
1: Fehler  
2: Daten-/Konvertierungsfehler
  - Alle Attributdefinitionen sind für die Elementkonvertierung ab dieser Version erforderlich, so muss jedes Attribut aus den zu konvertierenden Daten auch in der jeweiligen `ATTLIST`-Definition des betreffenden Elementes definiert sein. Im Konvertierungsmodus betrifft das nur die definierten Elemente.
- 1.21 14.11.2002
- Fehler bei Ausführung von `AFTER`, `NOTAFTER`, `BEFORE` und `NOTBEFORE` beseitigt (der folgende Befehl wurde in der Version 1.20 ignoriert).
  - Rekursionsschutz auf den `COPY`-Typ `DOC` beschränkt. Der Rekursionsschutz vergleicht die Kontextnamen der aufrufenden Routinen und bricht das Programm ab, wenn diese Bezeichnungen identisch sind.

- 1.20 30.8.2002
- AFTER, BEFORE, IN, INX, NOTAFTER, NOTBEFORE, NOTIN, NOTINX können jetzt mehrere Elemente mit einer Anweisung abfragen. Dazu können die Elementnamen mit Komma getrennt aufgeführt werden. Z.B.  
`IF IN p,liste,fn`
  - Attributabfragen zweier Attribute möglich. Z.B.  
`IF [id=refid]`
- 1.19 8.8.2002
- COPY/CUT mit dem Datenersetzungstyp `DROP` erweitert.
  - Korrektur der `END`-Regeln bei Verwendung in `COPY` oder `CUT`.
- Achtung!
- Der `COPY`- und `CUT`-Befehl führt unabhängig vom gewählten Ersetzungsmodell die Elementregeln des angegebenen Befehls stets aus. So lassen sich selbst beim Auslassen der Daten (`DROP`) Attribute vom angegebenen Element auslesen. Die in `COPY/CUT` angegebenen Datentypen können im Zielelement mit `DAT=...` überschrieben werden. (In den Vorversionen ging das nicht!)
- 1.18 1.8.2002
- Fehler bei der Entitätenkonvertierung innerhalb von Attributen beseitigt.
- 1.17 27.6.2002
- Parser erkennt jetzt ungültige Daten hinter der Instanz.
  - Parser prüft jetzt Zeichencodierung. Dabei werden Zeichen `>= 126` generell als ungültig betrachtet, im Steuerzeichenbereich werden nur Tabulator (`HT`), Zeilenvorschub (`LF`) und Wagenrücklauf (`CR`) erlaubt.
  - Probleme mit `DROP` beseitigt.
  - Alte Befehle `COPYALTALL`, `COPYALTONE`, `COPYASCON`, etc. funktionieren wieder wie original definiert.
- 1.16 6.6.2002
- Rekursionsschutz für die Befehle `COPY`, `CUT` und `DEL` hinzugefügt.
  - Skriptbefehle entsprechend der Groß-/Kleinschreibung (`XML`) für Benennungen der Elemente und Attribute umgestellt.

- 1.15      31.5.2002
- Probleme mit `COPY/CUT` und `DEL` beseitigt.
- HINWEIS: Im Element, dass mit `CUT` ausgeschnitten werden soll, darf am Endtag selbst kein `COPY`, `CUT` oder `DEL` stehen, da beim `CUT` die Daten sequentiell kopiert und gelöscht werden. Ein `COPY`, `CUT` oder `DEL` startet die Unterelementsuche stets vom Starttag des eigenen Elementes aus - und dieser Starttag wäre im gegebenen Fall bereits gelöscht worden. Alternative: Erst `COPY`-Befehl, dann `DEL`-Befehl ausführen.
- 1.14      27.5.2002
- `DAT=DROP` in allen Kontexten erlaubt.
  - `CUT` mit allen Ersetzungstypen wie `COPY` definiert.
  - `DEL` hinzugefügt (Typ `FULL` oder `DATA`).
- 1.13      16.5.2002
- `CUT` muss jetzt parametergesteuert angegeben werden:  
`FULL` = Alle Elemente und Daten ausschneiden (kopieren und entfernen (Achtung: Die Daten werden beim Parsen entfernt; Kontextregeln dürfen nicht auf bereits entfernte Elemente angewendet werden!))  
`DATA` = Nur die Daten ausschneiden (kopieren und entfernen), Elemente bleiben erhalten.
- 1.12      2.5.2002
- Problem mit Groß-/Kleinschreibung der Attributnamen im Skript beseitigt. Elemente und Attribute können XML-konfirm in der definierten Groß-/Kleinschreibung angegeben werden.
  - XML als Vorgabe definiert.
  - Probleme mit `CUT` beseitigt.
- 1.11      26.4.2002
- Probleme mit temporären Dateien beseitigt. MS Visual C++ Bug (Article ID: Q51326).
- 1.10 (b)    18.4.2002
- interne Programmänderung, Modul für SGML-Deklaration eingefügt. Steuerung von XML/SGML jetzt über dieses Modul.

- 1.10      11.4.2002
- Attributausgabe im Konvertiermodus für undefinierte Elemente eingefügt.
  - XML- und SGML-Empty-Elemente werden auch im Konvertiermodus korrekt bestimmt.
- 1.9        17.3.2002
- Attributabfragen verändert bzw. Vorgabewert-Auswertung korrigiert.
- Achtung! Zur Abfrage/Auswertung von Attributen muss im aktuellen Element nur das Attribut (`[Attribut]`) angegeben werden, ansonsten vollständig (`Element[Attribut]`). Eine Abfrage einer Eigenschaft (Attribut) eines übergeordneten Elementes ohne Angabe des Elementnamens ist nicht mehr möglich!

1.8

22.2.2002

- **BEG/END/DAT/ERR-Regelsätze** korrigiert. Funktionsweise geprüft.
- **COPY/CUT-Funktionsweise** korrigiert.
- Bei den Typen: **DATA** (Nur Daten komplett konvertieren, Keine Unterelemente), **ALT** (Unterelemente und Daten alternativ konvertieren), **ASC** (Unterelemente und Daten nach ASCII konvertieren), **DROP** (Daten und Unterelemente auslassen) werden die Regeln der aufgerufenen Element nicht mehr bearbeitet! Es werden nur die reinen Daten konvertiert bzw. ausgelassen
- **COPY/CUT-Funktionsweise** korrigiert. Die angegeben Ersetzungstypen (**FULL**, **ALTFULL**, **ASCFULL**, ...) werden auf alle Unterelemente vererbt. Achtung! Diese Ersetzungstypen können in den Regeln eines Unterelementes überschrieben werden.

+ Befehlsnotation der alten **COPY**-Befehle bereinigt:

```

NEU:           ALT
-----
COPYONE:       FINDONE
COPYALL:       FINDALL
COPYALTONE:    ALTONE
COPYALTALL:    ALTALL
COPYASCONE:    ASCONE
COPYASCALL:    ASCALL
    
```

- **Neue Schalter zur XML Convertersteuerung:**  
**CONVERTELEMENTS = ALL|ONLYDEFINED\*\***  
**CONVERTENTITIES = ALL|ONLYDEFINED**

Die Steuerung bewirkt:

**ALL:** Alle Elemente oder Entitäten werden übersetzt, d.h., für jedes Element oder jede Entität muss eine Ersetzungsregel existieren.

**ONLYDEFINED:** Nur die definierten Elemente oder Entitäten werden übersetzt, alle nicht definierten Elemente oder Entitäten bleiben im Zielfile erhalten.

\*\* Diese Funktion ist nur im XML-Mode verfügbar!

## 1.7

15.2.2002

- **COPY/CUT/SUB** jetzt parametergesteuert über Kontextnamen. Den Befehlen `COPY` und `CUT` kann als letzter Parameter ein frei definierbarer Kontextbezeichner mitgegeben werden. Dieser Parameter kann in den `SUB`-Abfragen ausgewertet werden, wodurch eindeutige Kontextzuordnungen ermöglicht werden. Neue Syntax:

```
COPY (Elementname, Suchtyp, Ersetzungstyp, Kontextname?)
```

```
CUT (Elementname, Suchtyp, Kontextname?)
```

```
SUB
```

```
SUB (Kontextname)
```

Wenn kein Kontextname angegeben wird, wird der aktuelle Elementname als Kontextname verwendet.

**Beispiel:**

Im Element `dok` sollen alle untergeordneten Elemente `index` zuerst als Attributwerte eingesammelt und mit `" | "` getrennt werden, dann sollen alle Elemente `"index"` als Liste dargestellt werden:

```
ELEMENT dok
BEG='<HTML><HEAD>'
BEG='<...
keywords="' + COPY(index, ALL, ASC, sammeln) + "'></
HEAD>'
BEG='<BODY><DIV>' + COPY(index, ALL, FULL, liste)
+'</DIV></BODY>'
END='</HTML>'
...
ELEMENT index // (fuer Kontext sammeln)
IF SUB(sammeln) // Alle Stichworteintraege
  IF NOTFIRST // suchen und mit
  Trennzeichen
    BEG=" | " // formatieren.
  ENDIF // (fuer Kontext liste)
ELSEIF SUB(liste) // Alle Stichworteintraege
  BEG="<P>" // suchen und als Textliste
  END="</P>" // formatieren.
ENDIF
```

1.6

7.2.2002

- Syntax bereinigt, Kontextregeln korrigiert.
- Probleme mit "aufhängendem" NOTFIND behoben.
- **Neue Syntax** für COPY, CUT, FIND, NOTFIND, TOPLEVEL, NOTTOPLEVEL
- TOPLEVEL und NOTTOPLEVEL dürfen nicht mehr in Klammern angegeben werden

```
ALT: IF (TOPLEVEL) ...
```

```
NEU: IF TOPLEVEL ...
```

- FIND und NOTFIND dürfen nicht mehr in Klammern angegeben werden

```
ALT: IF (FIND(p)) ...
```

```
NEU: IF FIND(p) ...
```

- Die erweiterten Kontextabfragen in FIND und NOTFIND dürfen nicht mehr separat in Klammern angegeben werden

```
ALT: IF (FIND((p IN eintrag))) ...
```

```
NEU: IF FIND(p IN eintrag) ...
```

- FIND und NOTFIND erhalten einen optionalen Parameter, den Suchtyp (DOC oder ONE)

DOC: Suche im gesamten Dokument

ONE: Suche ein Unterelement

**Beispiel 1:**

```
NEU: IF FIND(fn,DOC) ...
```

ermittelt das Vorhandensein von Fußnoten im gesamten Dokument

**Beispiel 2:**

```
NEU: IF FIND(fn) ...
```

ermittelt das Vorhandensein von Fußnoten innerhalb von Unterelementen des aktuellen Elementes.

- COPY und CUT erhalten einen neuen Parameterwert für den Suchtyp (DOC)

DOC: Suche und Bearbeitung aller Elemente im gesamten Dokument und die bisherigen Werte

ONE: Suche und Bearbeitung ein Unterelement

ALL: Suche und Bearbeitung aller Unterelemente.

- 1.5 31.1.2002
- **Bedingungstypen erweitert:** BEFORE und NOTBEFORE, AFTER und NOTAFTER, FIRST und NOTFIRST, LAST und NOTLAST, IN und NOTIN, INX und NOTINX, TOPLEVEL und NOTTOPELVEL, sowie FIND und NOTFIND.
  - **Beliebige Verschachtelungen von Kontextbefehlen jetzt möglich** (IN, NOTIN, INX, NOTINX, BEFORE, NOTBEFORE, AFTER, NOTAFTER, FIRST, NOTFIRST, LAST und NOTLAST).
- 1.4 24.1.2002
- Kommentare wurden realisiert.
  - Kommentarzeilen werden mit // eingeleitet und mit einem Zeilenwechsel abgeschlossen.
  - Kommentarbereiche werden durch /\* eingeleitet und mit einem \*/ abgeschlossen.
  - **Achtung!** Kommentare dürfen nur in Whitespace-Bereichen des Skriptes eingefügt werden, z.B. korrekt:
- ```
BEG=CHR(24) // Kommentar  
BEG=/*Kommentar*/CHR(24)
```
- 1.3B 17.1.2002
- Die Befehle BEG/END/ERR wurden um den Ersetzungsmodell-Befehl DAT erweitert.
  - Im COPY/CUT-Befehl müssen jetzt alle Parameter stets angegeben werden. Der Ersetzungstyp ist dem DAT-Befehl angepasst worden.
  - Die Ersetzungsmodell-Varianten für den Befehl DAT sowie für die COPY-Anweisung wurden wie folgt erweitert: FULL (Unterelemente und Daten komplett konvertieren), DATA (Nur Daten komplett konvertieren, Keine Unterelemente), ALT (Unterelemente und Daten alternativ konvertieren), ALTFULL (Nur Daten alternativ konvertieren, Keine Unterelemente), ASC (Unterelemente und Daten nach ASCII konvertieren), ASCFULL (Nur Daten nach ASCII konvertieren, Keine Unterelemente) und DROP (Daten und Unterelemente auslassen), wobei zu beachten ist, dass auch bei den Typen DATA, ALT und ASC die Daten der Unterelemente mit konvertiert werden.

- 1.3A 10.1.2002
- Bei der `TOKEN`-Verarbeitung werden jetzt Fehler lokal (in den Daten und im Skript) angezeigt.
  - Fehler bei Einheitenumrechnung ohne Angabe der Ausgangseinheit beseitigt bzw. Fehlermeldung generiert.
  - Bei `COPY/CUT` wurde im Debugmodus eine Fehlermeldung bei fehlendem Treffer zur Kontrolle generiert.
- 1.2E 19.7.2001
- Problem der `COUNT`-Funktion der Version 1.02D behoben.
- 1.2D 5.7.2001
- Die Kontextabfragen `IN` und `INX` wurden korrigiert. So wurde ein Bug der ersten Version beseitigt, der in der Kontextabfrage das eigene Element mit berücksichtigt hatte. Hinweis: Skripte, die diese alte Funktionalität benötigen, müssen angepasst werden.
- 1.2C 2.5.2000
- Fehlermeldung bei nicht angegebenen Skriptnamen hinzugefügt.
  - Fehlermeldung bei abgelaufener Lizenz hinzugefügt.
  - XML-Namensbereich mit ":" hinzugefügt.
- 1.2B 26.4.2000
- Generelle Änderung des Parameters `DTD`; dieser wird durch die Parameter `SYSTEM` und `PUBLIC` ersetzt.
  - Es kann ein System-Identifizier (`SYSTEM`) für das Skript definiert werden und zusätzlich (optional) ein oder mehrere Public-Identifizier (`PUBLIC`).
- Beispiel:
- ```
SYSTEM = "test.dtd"
PUBLIC = "-//Musterfirma//Test 1.00//DE"
PUBLIC = "-//Musterfirma//Test 1.01//DE"
```
- Die eingetragenen `SYSTEM`- bzw. `PUBLIC`- Identifizier werden beim Parsen der Instanzen geprüft. Wird ein nicht definierter Identifizier verwendet, bricht der Parser mit einer Fehlermeldung ab.
- Zweite ausgelieferte kommerzielle Version.**

- 1.2A 19.4.2000
- **SYNTAX**-Schalter im Header (**SGML** oder **XML**) hinzugefügt
  - **XML** Unterstützung der **Empty-Tags**
  - **MODEL**-Definition optional eingestellt, Vorgabewert **MIXED**
  - Fehlerbehandlung bei nicht definierten Elementen oder Entitäten tolerant gestaltet (Fehlermeldung, aber kein Abbruch mehr)
  - Erweiterter Namensbereich um "-" und "\_" für den Parser festgelegt
- 1.1A 13.4.2000
- Debugmodus mit Kommandozeilenschalter "/d" hinzugefügt.
  - Debugmeldungen im Skript werden mit Zeilen- und Zeichennummer angegeben.
  - Problem mit Abfragen der Vorgabewerte von Attributen in übergeordneten Elementen behoben
- 1.0J 28.11.1999
- Problembeseitigungen und Anpassungen für den Einsatz mit Windows NT.
  - Für die Ausgabe können jetzt relative Pfade angegeben werden.
  - Ausgabe von Anwendungsfehlermeldungen **ERR** mit Positionsangabe.
- 1.0I 9.11.1999
- Problem bei der Bearbeitung von **COPY/CUT** mit erweiterten Kontextbedingungen beseitigt.
- 1.0H 2.7.1999
- **TOKEN**: Fehler bei Summenberechnung ohne Angabe eines Korrekturfaktors oder einer neuen Einheit beseitigt!
- 1.0G 20.6.1999
- **TOKEN**: Es werden generell keine Einheiten mehr zurückgegeben. Wenn keine Umrechnung erfolgen soll, wird trotzdem die alte Einheit abgeschnitten!
  - **TOKEN**: positiver/negativer Offset als Suffixparameter ": + {wert}" bzw. ":-{wert}" realisiert.

- 1.0F 12.6.1999
- Für die `TOKEN`-Funktion wurde eine spezielle Unterfunktion integriert, mit der die korrigierte Gesamtsumme aller Tokenwerte berechnet werden kann. Dazu muss bei der Einheit der Präfix "s:" angegeben werden.
- 1.0E 3.6.1999
- Erweiterte Fehlermeldung für `COUNT`
  - Geänderte Struktur für Start- und Endtags mit erweiterten Fehlermeldungen
  - Erweiterte Fehlermeldungen bei `VALUE`, `ASCVALUE` und `ALTVALUE`
  - Bedingung `SUB` hinzugefügt.
  - `COPY` und `CUT` haben eine gemeinsame Funktion bekommen: `COPYCUT`, die mit dem Parameter `CUTDATA` gesteuert wird
  - `FULL` und `ALTFULL` überarbeitet. Mit der neuen Condition `SUB` lässt sich mit normalen Elementregeln das Ersetzungsverhalten steuern.
  - Im Parser wurden Fehlermeldungen für fehlerhaft schließende Elemente verbessert.
- 1.0D 27.5.1999
- `ALT-` und `ASC-VALUE` hinzugefügt
  - `FILENAME` hinzugefügt zur Erzeugung einer einzigen Ausgabe-datei bei mehreren Eingabedateien.
  - `COPY` und `CUT` kann per Parametersteuerung jetzt wahlweise alle (`ALL`) oder nur den ersten Treffer (`ONE`) umsetzen
- 1.0A 18.5.1999
- 1.0B
- 1.0C
- `ASCII`-Ersetzung hinzugefügt.
  - `TOKEN` erweitert (%).
  - `ASCALL` und `ASCONE` als zusätzliche `ASCII`-Funktionen hinzugefügt.
  - `EMPTY`-Bedingung hinzugefügt.
- 1.0 13.4.1999
- Erste ausgelieferte kommerzielle Version.**  
**Produktname: „mediaTEXT SGML Konverter“**



START      28.1.1999  
Beginn des Projektes  
Bis zur Auslieferung der Version 1.00 wurden insgesamt 69  
Revisionen erstellt.

# Quellen

## Literatur

---

[CHARSETS]	IANA Character Sets Official names for character sets. <a href="http://www.iana.org/assignments/character-sets">www.iana.org/assignments/character-sets</a>
[CSS]	W3C Cascading Style Sheets <a href="http://www.w3.org/Style/CSS/">www.w3.org/Style/CSS/</a>
[HTML]	HTML5 A vocabulary and associated APIs for HTML and XHTML W3C Recommendation 28 October 2014 <a href="http://www.w3.org/TR/2014/REC-html5-20141028/">www.w3.org/TR/2014/REC-html5-20141028/</a>
[SGML]	SGML Syntax Summary Table of Contents <a href="http://xml.coverpages.org/sgmlsyn/contents.htm">xml.coverpages.org/sgmlsyn/contents.htm</a>
[UCS-2]	siehe auch Unicode 1.0 <a href="http://www.unicode.org/versions/Unicode1.0.0/">http://www.unicode.org/versions/Unicode1.0.0/</a>
[UNICODE]	The Unicode Standard <a href="http://www.unicode.org/standard/standard.html">www.unicode.org/standard/standard.html</a>
[Web SGML]	ISO 8879 TC2, Annex K Web SGML Adaptations <a href="http://www.sgmlsource.com/8879/n0029.htm">www.sgmlsource.com/8879/n0029.htm</a>
[XML]	Extensible Markup Language (XML) 1.0 (Fifth Edition) W3C Recommendation 26 November 2008 <a href="http://www.w3.org/TR/xml/">www.w3.org/TR/xml/</a>

## Editoren

---

*Editoren mit integrierter Unterstützung für den XML Converter:*

[PSPad]	PSPad Freeware Editor von Jan Fiala Version 5.0.3 (377) vom 09.02.2020 <a href="http://www.pspad.com">www.pspad.com</a>
---------	--

*Weitere anpassbare Editoren:*

[BBEdit]	BBEdit für Mac OS X 10.9.5 oder neuer. Version 11.6.2 (397016) vom 15.09.2016 <a href="http://www.barebones.com/products/bbedit/">www.barebones.com/products/bbedit/</a>
[Brackets]	Brackets Release 1.3 vom 11.05.2015 für Linux, Mac OS X und Windows <a href="https://github.com/adobe/brackets">github.com/adobe/brackets</a>
[CodeMirror]	CodeMirror für Browser

---

	Version 5.3 vom 20.05.2015 <a href="http://codemirror.net">codemirror.net</a>
[Codepad2]	Codepad Texteditor für Windows Version 2.0.4.0 vom 30.10.2011 <a href="http://www.h84.net">www.h84.net</a>
[CodePad4]	CodePad code and hex editor für Windows Version 4.1 vom 16.12.2006 <a href="http://shicola.wz.cz/data/codepad/CodePad.html">shicola.wz.cz/data/codepad/CodePad.html</a>
[Fraise]	Fraise für Mac OS X 10.6 und neuer. Version 3.7.3 vom 29.3.2010 <a href="https://github.com/jfmoy/Fraise">github.com/jfmoy/Fraise</a>
[NotePad++]	NotePad++ Version 7.6.6 vom 04.04.2019 <a href="https://notepad-plus-plus.org/download/">https://notepad-plus-plus.org/download/</a>
[TextPad]	TextPad Text Editor für Windows Version 8.4.0 vom 17.04.2020 <a href="http://www.textpad.com/">www.textpad.com/</a>
[TextWrangler]	TextWrangler für Mac OS X 10.9.5 und neuer. Version 5.5.2 (3786) vom 20.09.2016 <a href="http://www.barebones.com/products/textwrangler/">www.barebones.com/products/textwrangler/</a> Mac App Store <a href="https://itunes.apple.com/de/app/textwrangler/id404010395#">itunes.apple.com/de/app/textwrangler/id404010395#</a>
[Tincta]	Tincta für Mac OS X 10.8 oder neuer. Version 4.0.1 (4010) vom 23.02.2020 Mac App Store <a href="https://itunes.apple.com/am/app/tincta/id448340648#">itunes.apple.com/am/app/tincta/id448340648#</a>
[UEStudio]	UE Studio für Windows Version 19.00 vom 29.03.2019 <a href="https://www.ultraedit.com/downloads/uestudio-download/">https://www.ultraedit.com/downloads/uestudio-download/</a>

[UltraEdit]

UltraEdit Text Editor für Windows

Version 27.00 vom 19.05.2020

<https://www.ultraedit.com/downloads/ultraedit-download/>

UltraEdit Text Editor für macOS

(für Mac OS X 10.12 oder neuer)

Version 20.00 vom 20.12.2019

<https://www.ultraedit.com/downloads/ultraedit-download/>

UltraEdit Text Editor für Linux

Version 18.00 vom 31.07.2018

<https://www.ultraedit.com/downloads/ultraedit-download/>