

XML Converter

Handbuch und Referenz

Michael Seume



Lizenzbestimmungen:

Copyright © 2008, 2009 Michael Seume

Die Software und die Dokumentation sind urheberrechtlich geschützt.

Die Software und die Dokumentation werden vom Autor unentgeltlich und ausschließlich über "www.xml-converter.de" zur Verfügung gestellt.

Hiermit wird unentgeltlich jeder Person, die eine Kopie der Software und der Dokumentationen erhält, die Erlaubnis erteilt, diese uneingeschränkt zu benutzen. Das Kopieren, Ändern, Fusionieren, Verlegen, Verbreiten, Unterlizenzieren und/oder Verkaufen ist untersagt. Die Weitergabe bedarf der ausdrücklichen Genehmigung durch den Lizenzgeber.

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.

Impressum:



XML Converter

(XMLcnv.exe)

Software vom 07.03.2009 (Version 1.27.2)

Handbuch vom 07.03.2009

© 2008, 2009 Michael Seume,
1999–2007 mediaTEXT Jena GmbH

Alle Rechte vorbehalten!

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

Inhaltsverzeichnis

Vorwort.....	7
Skriptsprache und Parser.....	8
Wohlgeformtes SGML.....	11
Wohlgeformtes XML.....	14
Benutzung des Handbuches – Konventionen	17
Bedienung	19
Programmierung	22
Prinzipielle Arbeitsweise.....	22
So funktioniert ein Skript	24
Wir erstellen ein Skript	27
Optimierung des Skriptes.....	29
Sprachbeschreibung	31
Anweisung	32
Attributbedingung.....	33
Bedingung	35
Bedingungsausdruck.....	36
Element-Ausgabeanweisung	37
Element-Handler.....	38
Element-Konvertierungsregeln	39
Entitäten-Ausgabeanweisung.....	40
Entitäten-Handler	41
Entitäten-Konvertierungsregeln	42
Kommentar	43
Kontextbedingung	44
Skript	46
Skript-Einstellungen	48
Unterelement	49
Unterstruktur	50
Verzweigung.....	51
Zeichenkette	53
Zeichenkette, eingeschränkte.....	54
Sprachreferenz.....	57
" "	57
' '	59

[].....	61
//	64
/* */	65
AFTER.....	67
ALTERNATE.....	71
ASCII.....	73
ATTLIST.....	75
BASE.....	77
BEFORE.....	79
BEG.....	83
CHR	87
CONVERTDATA.....	88
CONVERTELEMENTS	90
CONVERTENTITIES	92
COPY	93
COUNT.....	97
CUT	101
DAT	104
DEL	107
ELEMENT.....	109
ELSE	112
ELSEIF.....	113
EMPTY	114
END.....	116
ENDE	120
ENDIF.....	121
ENTITY.....	122
ERR.....	124
FILE.....	126
FIND.....	129
FIRST	132
IF	134
IMPLIED.....	135
IN	142
INX	146
LAST	150
MODEL.....	152
NOT.....	154
PUBLIC	157
SUB.....	158
SYNTAX.....	161

SYSTEM.....	162
TOKEN.....	164
TOPLEVEL.....	168
VALUE.....	170
Fehlermeldungen	177
XML/SGML-Glossar	178
ATTLIST.....	178
Attribut.....	181
Attributname	182
Attributwert	183
Ausführungsanweisung.....	184
Ausschluss	185
Bezeichner.....	186
BOM.....	188
CDATA	189
DOCTYPE	190
Dokument.....	192
DTD.....	194
Einschluss	195
ELEMENT.....	196
Element	198
Elementname.....	199
ENTITY.....	200
Entität	204
Kommentar	206
Entitätenname.....	207
Instanz.....	208
Leerraum	209
NDATA	210
NAMECASE	211
NOTATION.....	212
OMITTAG	213
PCDATA.....	215
Prolog.....	216
PUBLIC	217
RCDATA.....	218
SDATA	219
SGML-Deklaration	220
SHORTTAG.....	222
SUBDOC	224

SYSTEM.....	225
Vereinbarung	226
Vereinbarung, erweiterte	227
Vereinbarung, leere	228
XML-Deklaration	229
Zeichenverweis.....	231
Kompatibilitätsmodus	233
Übersicht über die Änderungen	233
Aufruf.....	236
Attributbedingung []	238
Skript-Einstellungen	240
ALTVALUE	241
ASCVALUE	243
BASE.....	245
COPY	246
COPYALL.....	247
COPYALTALL.....	248
COPYALTONE	249
COPYASCALL.....	251
COPYASCONC	253
COPYONE.....	255
COUNT.....	257
COUNTER.....	257
CUT	259
EXTENSION.....	259
FILENAME.....	261
OWNER.....	262
Editoren.....	263
CodePad Konfiguration	263
NotePad++ Konfiguration.....	264
PSPad Konfiguration.....	265
TextPad Konfiguration.....	267
UltraEdit Konfiguration	268
Systemanforderungen	269
Entwicklungsgeschichte	273
Quellen	292
Literatur	292
Editoren	292



Vorwort

Die Entwicklung dieses Werkzeuges startete im Januar 1999. Es bestand zu dieser Zeit die Notwendigkeit, ein einfach handhabbares Werkzeug zur Medienaufbereitung von SGML-Daten zu schaffen. Das Ziel war eine einfache Sprache zur kontextbasierten Auswertung und Bearbeitung von SGML-Daten. Und dieser Kernbereich - die Konvertierung der Daten mittels Strukturkonvertierung wurde konsequent weiterentwickelt. Textuelle Manipulationen innerhalb der Daten (Veränderungen und Korrekturen der #PCDATA-Bereiche in den Daten) wurden anderen Werkzeugen überlassen. Für diesen Zweck gab es bereits ausreichend Bearbeitungswerkzeuge.

Bei der Entwicklung wurden aber auch Anforderungen an die Daten definiert und implementiert, die die Bearbeitung und Kontrolle von SGML-Daten in dieser Zeit deutlich vereinfachten (siehe Wohlgeformtes SGML).

Basierend auf diesen SGML Grundlagen konnte der Umstieg auf XML bei Beibehaltung dieser Systematik vollzogen werden. So kann mit diesem Werkzeug (XML Converter) sowohl SGML als auch XML bearbeitet werden (siehe Wohlgeformtes XML).

Typische Anwendungen sind:

- XML-HTML Konvertierung
- XML-CSV Konvertierung
(CSV: Comma Separated Values)
- XML-DOC Konvertierung
(DOC: Worddokumente via HTML oder WordML)
- XML-FFF Konvertierung
(FFF: Folio Flat File)
- XML Datenstrukturprüfung
- SGML-HTML Konvertierung
- SGML-CSV Konvertierung
- SGML-DOC Konvertierung
- SGML-FFF Konvertierung
- SGML Datenstrukturprüfung

Skriptsprache und Parser

Dieses Werkzeug (XML Converter) besteht aus einem XML-/SGML-Parser und einer integrierten Skriptsprache.

Der Parser liest XML-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>  
</buch>
```

Wohlgeformtes SGML

SGML Daten zu verarbeiten, erschien 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 `ß` wie folgt für den Parser korrekt eingesetzt werden: "Ich weiß es" aber auch "Ich weiß 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. "§ 11" statt "§; 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 "&" oder "&" ausgezeichnet werden.
- Das "<"-Zeichen ist als Textzeichen nicht erlaubt und muss als Entität "<" oder "<" ausgezeichnet werden.
- Das ">"-Zeichen ist als Textzeichen nicht erlaubt und muss als Entität ">" oder ">" 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 wer-

den 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 ">", ">" oder ">" 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:

1. ANSI, ASCII, US-ASCII,
2. ISO-8859-1 bis ISO 8859-15 und
3. 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.

Benutzung des Handbuches – 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 angehängten 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 Converter ist ein 32bit-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.exe eing skript ausg /s /w /o /l /c /d
```

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 "
```

oder mit Wildcards

```
"C:\Test\*.xml "
```

oder

```
"C:\Test\TE?T.xml "
```

oder relative Pfadangabe

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

oder indirekte, relative Pfadangabe

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

skript Projektname. Es genügt der Skriptname ohne Erweiterung (z.B. "Html"). Die Projektdatei sollte sich im gleichen Ver-

zeichnis wie das Programm befinden. Die Standarderweiterung ist ".xb";

Beispiel:

"C:\Test\Html"

oder

"C:\Test\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.

Beispiele:

Vollständige Dateibenennung des Ergebnisses:

"C:\Test\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.

- /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)
- /c Kompatibilitätsmodus

Der Erfolg der Programmausführung wird durch den Rückgabewert (Errorlevel) angezeigt (siehe Fehlermeldungen).

Beispiel 1

```
XMLcnv.exe C:\Test\Doku.xml Html C:\Test\DOKU.htm
```

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

Beispiel 2

```
XMLcnv.exe 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 3

```
XMLcnv.exe C:\Test\*.xml Html C:\Test\Geamt.htm /s/o/lg
```

Konvertiere alle XML-Dateien im Verzeichnis `C:\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 Kon-
-----------	--

vertierungsregeln 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 im **UTF-8** Modus. Wenn Sie ein Skript für SGML erstellen, verwenden Sie bitte eine Texteditor im "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-Konvertierungsregeln* 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*.

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.

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-Ausgabeeinweisung*
- *Entitäten-Ausgabeeinweisung*

Attributbedingung

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

Für die Abfrage eines vererbten Attributwertes muss **IMPLIED** verwendet werden.

Syntax

[*Attributname* *Operator* *Zeichenkette*]

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)

Zeichenkette *Zeichenkette* mit einem Vergleichswert.

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

Bedingungsausdruck

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

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-Konvertierungsregeln

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

Syntax

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

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

Siehe auch

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

Entitäten-Ausgabebezeichnung

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

Syntax

{Zeichenkette|VALUE|CHR}

Zeichenkette Ein an dieser Position auszugebender Text.

Hinweis

Rekursionsschutz! Bei Verwendung von *VALUE* innerhalb einer *Entitäten-Ausgabebezeichnung* 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-Konvertierungsregeln***

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 Zeilenwechseln bestehen. Auch innerhalb von Anweisungen sind Whitespace-Bereiche entsprechend der Anweisungssyntax zulässig.

Syntax 1

`//` Kommentarzeile

<i>Kommentar</i>	Kommentarzeile, bestehend aus beliebigen Text, der mit einem Zeilenwechsel (Enter, CR LF) abgeschlossen wird.
------------------	---

Syntax 2

`/*` Kommentarbereich `*/`

<code>/*</code>	Beginn des Kommentarbereiches.
<i>Kommentarbereich</i>	beliebiger Textbereich. Dieser Bereich kann sich auch über mehrere Zeilen erstrecken.
<code>*/</code>	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.

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 Whitespaces-Bereichen zulässig. Whitespace-Bereiche sind Trennbereiche zwischen Anweisungen, die in der Regel aus Leerzeichen, Tabulato-

ren und Zeilenwechseln 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 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} |
{CONVERTELEMENTS} |
{CONVERTENTITIES} |
{CONVERTDATA} |
{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-Konvertierungsregeln*. Eine Verschachtelung von Verzweigungen ist in beliebiger Tiefe möglich.

Syntax

```
IF Bedingungsausdruck  
  Element-Konvertierungsregeln  
{ELSEIF Bedingungsausdruck  
  Element-Konvertierungsregeln}*  
{ELSE  
  Element-Konvertierungsregeln?  
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-Konvertierungsregeln*

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 die-

se 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).

Hinweis

Wenn Sie im Kompatibilitätsmodus arbeiten, beachten Sie bitte die Hinweise im Abschnitt *Attributbedingung* [].

Syntaxbeispiele

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

Syntax 1

[*Attributname*]

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

Syntax 2

[*Attributname*=*Vorgabewert*]

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

Vorgabewert *Zeichenkette*, die den Standard-*Attributwert* definiert. Der Standardwert wird immer dann verwendet, wenn innerhalb der Instanz kein Wert für das Attribut angegeben wurde.

Diese Syntax wird innerhalb der **ATTLIST**-Definition verwendet.

Syntax 3

[*Attributname* *Operator* *Wert*]

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

<i>Operator</i>	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*.

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 [id=""]
```



Dies ist eine Bedingung, die prüft, ob das *Attribut* id leer ist. Hinweis: Vergleichsoperationen werden als Textvergleiche durchgeführt.

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 beliebigen 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 *Attributbe-*
dingung.

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 beitrag
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, TOPLEVEL, *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 = *Ausgabeeweisung* {+ *Ausgabeeweisung* }*

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.

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 `ü` konvertiert. Bei einem alternativen Ersetzungsmodell wird diese Entität dann alternativ mit `ue` umschrieben.

Hinweis

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

Siehe auch

ENTITY, **ASCII**, *Entitäten-Handler*, *Entitäten-Konvertierungsregeln*

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 = *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 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 `ü` konvertiert. Bei einem alternativen ASCII-Ersetzungsmodell wird diese Entität dann mit `ü` umschrieben.

Hinweis

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

Siehe auch

`ENTITY`, `ALTERNATE`, *Entitäten-Handler*, *Entitäten-Konvertierungsregeln*

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 f_n) 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*

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-Ausgabeeanweisung*, *Element-Konvertierungsregeln*, *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 (0-255).

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 ©.

Siehe auch

BEG, CHR, COPY, COUNT, CUT, DEL, END, ERR, FILE, IMPLIED, TOKEN, VALUE, *Element-Ausgabeeinweisung*, *Entitäten-Ausgabeeinweisung*, *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. (Standard)
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.

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. (Standard)
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-Konvertierungsregeln* konvertiert.

Beispiel 2

```
CONVERTELEMENTS=ONLYDEFINED
```

Es werden von den in den Daten vorkommenden Elementen nur diejenigen, die in den *Element-Konvertierungsregeln* 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-Konvertierungsregeln* 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. (Standard)

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-Konvertierungsregeln* konvertiert.

Beispiel 2

```
CONVERTENTITIES=ONLYDEFINED
```

Es werden von den in den Daten vorkommenden Entitäten nur diejenigen, die in den *Entitäten-Konvertierungsregeln* 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 und gibt die Daten anhand der angegebenen Ersetzungsregeln zurück. Die Daten werden an der aktuellen Position eingefügt.

Syntax 1

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

Syntax 2

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

Syntax 3

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

<i>Elementname</i>	<p><i>Bezeichner</i> für ein <i>Element</i>.</p> <p>Ohne Angabe eines Elementnamens wird das aktuelle Element verwendet. Achtung, in diesem Fall ist die Verwendung eines Kontextnamens zu empfehlen, um eine Rekursion zu vermeiden!</p>
<i>Attributbedingung</i>	<p>[<i>Attributname</i> <i>Operator</i> <i>Zeichenkette</i>]</p> <p><i>Bedingung</i>, die einen Wert eines <i>Attributs</i> abfragt. Syntax hierfür siehe <i>Attributbedingung</i></p>
<i>Kontextbedingung</i>	<p>eine <i>IN</i>, <i>INX</i> oder deren Negation mit dem NOT-Präfix.</p>
<i>Suchtyp</i>	<p>Gibt an, ob ein <i>Unterelement</i>, alle <i>Unterelemente</i> oder alle betreffenden Elemente im Dokument gesucht werden sollen:</p> <p>ONE Suche und Bearbeitung eines <i>Unterelementes</i> (erster Treffer),</p>

	ALL	Suche und Bearbeitung aller Unter- elemente ,
	DOC	Suche und Bearbeitung aller Elemente im gesamten Dokument.
<i>Ersetzungstyp</i>		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.
	ALT	Wie DATA, aber Entitäten werden alternativ umgesetzt.
	ASC	Wie DATA, aber Entitäten werden nach ASCII umgesetzt.
<i>Kontextname</i>		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.
<i>Trenntext</i>		Eine Zeichenkette , die als Abtrennung zwischen die einzelnen gefundenen Elemente eingefügt wird.

Der COPY/CUT-Befehl führt unabhängig vom gewählten Ersetzungsmodell die Elementregeln des angegebenen Elementes stets aus. So lassen sich selbst beim Auslassen der Daten (DROP) Attribute vom angegebenen Element auslesen.

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.

Siehe auch

BEG, CHR, COUNT, CUT, DEL, END, FILE, IMPLIED, TOKEN, VALUE, *Element-Konvertierungsregeln*, *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 Suchtyp angegeben, werden alle Vorkommen dieses Elementes auf der Ebene des gefundenen Elementes gezählt. Der Ergebniswert wird an der aktuellen Position eingefügt.

Syntax

COUNT(*{Elementname}*?*{,Suchtyp}*?*{,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

Suchtyp *{Suchbereich}{+Zählart}*?

Suchbereich:

LEVEL Sucht und zählt die Elemente auf der Ebene des gefundenen Elementes.

DOC Sucht und zählt alle Elemente im Dokument.

Vorgabe ist LEVEL+POS.

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()           // aktuelle Element zaehlen
BEG=COUNT(,ROM)        // roemisch zaehlen
BEG=COUNT(,ALPHA       // alphanumerisch zaehlen
BEG=COUNT(eintrag)     // 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-Konvertierungsregeln*, *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 Daten werden an der aktuellen Position eingefügt.

Syntax 1

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

Syntax 2

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

<i>Elementname</i>	<i>Bezeichner</i> für ein <i>Element</i> .						
<i>Attributbedingung</i>	[<i>Attributname</i> <i>Operator</i> <i>Zeichenkette</i>] <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 Negation mit dem <i>NOT</i> -Präfix.						
<i>Suchtyp</i>	Gibt an, ob ein <i>Unterelement</i> , alle <i>Unterelemente</i> oder alle betreffenden Elemente im Dokument gesucht werden sollen: <table> <tr> <td>ONE</td><td>Suche und Bearbeitung eines <i>Unterelementes</i> (erster Treffer),</td></tr> <tr> <td>ALL</td><td>Suche und Bearbeitung aller <i>Unterelemente</i>,</td></tr> <tr> <td>DOC</td><td>Suche und Bearbeitung aller Elemente im gesamten Dokument.</td></tr> </table>	ONE	Suche und Bearbeitung eines <i>Unterelementes</i> (erster Treffer),	ALL	Suche und Bearbeitung aller <i>Unterelemente</i> ,	DOC	Suche und Bearbeitung aller Elemente im gesamten Dokument.
ONE	Suche und Bearbeitung eines <i>Unterelementes</i> (erster Treffer),						
ALL	Suche und Bearbeitung aller <i>Unterelemente</i> ,						
DOC	Suche und Bearbeitung aller Elemente im gesamten Dokument.						
<i>Ersetzungstyp</i>	Gibt den Ersetzungstyp für die Daten an. Es stehen folgende Varianten zur Auswahl: <table> <tr> <td>FULL</td><td>Bearbeitet alle <i>Unterstrukturen</i> gemäß dem Skript. Die Start- und End-</td></tr> </table>	FULL	Bearbeitet alle <i>Unterstrukturen</i> gemäß dem Skript. Die Start- und End-				
FULL	Bearbeitet alle <i>Unterstrukturen</i> gemäß dem Skript. Die Start- und End-						

tags der Elemente und deren Inhalte werden nach dem Kopieren vollständig gelöscht!

DATA Setzt nur die Daten um, Elementregeln werden ausgelassen. Die Daten werden nach dem Kopieren gelöscht, die Start- und Endtags bleiben erhalten.

ALT Wie DATA, aber Entitäten werden alternativ umgesetzt. Die Daten werden nach dem Kopieren gelöscht, die Start- und Endtags bleiben erhalten.

ASC Wie DATA, aber Entitäten werden nach ASCII umgesetzt. Die Daten werden nach dem Kopieren gelöscht, die Start- und Endtags bleiben erhalten.

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 unabhängig vom gewählten Ersetzungsmodell die Elementregeln des angegebenen Elementes stets aus. So lassen sich selbst beim Auslassen der Daten (DROP) Attribute vom angegebenen Element auslesen.

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-Konvertierungsregeln*

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 eindex  
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-Konvertierungsregeln*, *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*}, *Suchtyp*, *Löschttyp*)

Syntax 3

DEL(*Elementname*{*Attributbedingung*}?{ *Kontextbedingung*}?,
Suchtyp, *Löschttyp*)

<i>Elementname</i>	<i>Bezeichner</i> für ein <i>Element</i> .						
<i>Attributbedingung</i>	[<i>Attributname</i> <i>Operator</i> <i>Zeichenkette</i>] <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 Negation mit dem <i>NOT</i> -Präfix.						
<i>Suchtyp</i>	Gibt an, ob ein <i>Unterelement</i> , alle <i>Unterelemente</i> oder alle betreffenden Elemente im Dokument gesucht werden sollen: <table> <tr> <td>ONE</td><td>Suche und Bearbeitung eines <i>Unterelementes</i> (erster Treffer),</td></tr> <tr> <td>ALL</td><td>Suche und Bearbeitung aller <i>Unterelemente</i>,</td></tr> <tr> <td>DOC</td><td>Suche und Bearbeitung aller Elemente im gesamten Dokument.</td></tr> </table>	ONE	Suche und Bearbeitung eines <i>Unterelementes</i> (erster Treffer),	ALL	Suche und Bearbeitung aller <i>Unterelemente</i> ,	DOC	Suche und Bearbeitung aller Elemente im gesamten Dokument.
ONE	Suche und Bearbeitung eines <i>Unterelementes</i> (erster Treffer),						
ALL	Suche und Bearbeitung aller <i>Unterelemente</i> ,						
DOC	Suche und Bearbeitung aller Elemente im gesamten Dokument.						
<i>Löschttyp</i>	Gibt den Löschttyp für die Daten an. Es stehen folgende Varianten zur Auswahl:						

FULL	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-Konvertierungsregeln*, *Zeichenkette*

ELEMENT

Definiert den Aufbau und die Konvertierung des angegebenen Elementes.

Syntax

ELEMENT *Elementname*

{MODEL}?

{ATTLIST}*

{*Element-Konvertierungsregeln*}*

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-Konvertierungsregeln 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-Konvertierungsregeln  
{ELSEIF Bedingungsausdruck  
  Element-Konvertierungsregeln}*  
{ELSE  
  Element-Konvertierungsregeln?  
ENDIF
```

Beschreibung

Siehe *Verzweigung*.

Siehe auch

IF, ELSEIF, ENDIF, *Bedingung*, *Bedingungsausdruck*, *Element-Konvertierungsregeln*, *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-Konvertierungsregeln  
{ELSEIF Bedingungsausdruck  
  Element-Konvertierungsregeln}*  
{ELSE  
  Element-Konvertierungsregeln?  
ENDIF
```

Beschreibung

Siehe *Verzweigung*.

Siehe auch

IF, ELSE, ENDIF, *Bedingung*, *Bedingungsausdruck*, *Element-Konvertierungsregeln*, *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 (` `) 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 = *Ausgabeeanweisung* {+ *Ausgabeeanweisung*}*

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

+ Verkettungsoperator. Mit diesem Operator werden Texte oder *Element-Ausgabeeanweisungen* 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-Ausgabeeanweisung*, *Element-Konvertierungsregeln*, *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-Konvertierungsregeln*}*

ENDE

Beschreibung

Siehe **ELEMENT**.

Siehe auch

ELEMENT, *Skript*, *Element-Handler*

ENDIF

Ende der *Verzweigung*.

Syntax

```
IF Bedingungsausdruck  
  Element-Konvertierungsregeln  
{ELSEIF Bedingungsausdruck  
  Element-Konvertierungsregeln}*  
{ELSE  
  Element-Konvertierungsregeln}?
```

ENDIF

Beschreibung

Siehe *Verzweigung*.

Siehe auch

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

ENTITY

Definiert die Entitäten-Ersetzung.

Syntax

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

Name gültiger XML-/SGML-Bezeichner für die Entität. Das Entitäten-Startzeichen (reference open) "&" und das Entitäten-Endezeichen (reference close) ";" sind wegzulassen.
Zeichenverweise (character references) werden mit der Voranstellung # angegeben.

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.

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-Ausgabeanweisung* 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"
```

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.

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-Konvertierungsregeln*, *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}?*)

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

Extension optionale Dateinamenserweiterung, die statt der Original-Dateinamenserweiterung der Datei zurückgegeben werden soll. Die Extension muss als *Zeichenkette* angegeben werden! Bei Angabe einer leeren *Zeichenkette*, 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 Entitätsdeklaration 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 Dateierweiterung auf `gif` geändert. Achtung: die Grafikdatei wird nicht umbenannt.



Siehe auch

BEG, CHR, COPY, COUNT, CUT, DEL, END, IMPLIED, TOKEN,
VALUE, *Element-Konvertierungsregeln*, *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*}?)

<i>Elementname</i>	Bezeichner für ein <i>Element</i> .
<i>Attributbedingung</i>	[<i>Attributname</i> <i>Operator</i> <i>Zeichenkette</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.

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`, `TOPLEVEL`, *Bedingungsausdruck*

IF

Leitet eine neue *Verzweigung* ein.

Wenn der *Bedingungsausdruck* wahr ist, werden die *Element-Konvertierungsregeln* 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-Konvertierungsregeln  
{ELSEIF Bedingungsausdruck  
  Element-Konvertierungsregeln}*  
{ELSE  
  Element-Konvertierungsregeln}?  
ENDIF
```

Beschreibung

Siehe *Verzweigung*.

Siehe auch

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

IMPLIED

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

IMPLIED-Anweisung

Ausgabe eines 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 einen positiven Offset

- numerischer Wert für einen negativen 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	0	12	5467,872	0,6785
#	0	12	5468	1
###	0,00	12,00	5467,87	0,68
###	000	012	5468	001

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

pt #.#

mm #.##

cm #.###

in #.####

% #.#

%% #.#

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

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=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-Konvertierungsregeln*, *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.

Bei Verwendung als Bedingung (Syntax 1) wird der Wahrheitswert zurückgegeben.

Bei Verwendung als Anweisung (Syntax 2) wird der Attributwert zurückgegeben.

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 angegebenen 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 be-

trachtet. 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, TOPLEVEL, *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 be-

trachtet. 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, TOPLEVEL, *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`, `TOPLEVEL`, *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*



NOT

Negation des folgenden Ausdrucks. Der Wahrheitswert des folgenden Befehls wird negiert.

Syntax

NOTAFTER

NOTBEFORE

NOTIN

NOTINX

NOTEMPTY

NOTFIND

NOTFIRST

NOTLAST

NOTEMPTY

NOTSUB

NOTTOPEVEL

Hinweis

Der Präfix NOT wird direkt an den zu negierenden Befehl angefügt. Es dürfen also keine Trennzeichen zwischen NOT und dem Befehl stehen:

- NOT AFTER (falsch!)
- **NOTAFTER** (richtig!).

Bei der Auswertung des Ausdruckes wird nur der verbundene Befehl negiert, 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*

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**

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 `fn`) an das Ende des Dokumentes. Innerhalb des Elementes `fn` 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`, `TOPELVEL`, *Bedingungsausdruck*



SYNTAX

Schaltet den internen Parser auf die gewünschte Syntax XML oder SGML um.

Syntax

SYNTAX = {XML|SGML}

XML Voreinstellung. XML-Syntax.

SGML SGML-Syntax gemäß Reference Concrete Syntax.

Beispiel 1

SYNTAX=XML

Es soll die XML-Syntax verwendet werden.

Beispiel 2

SYNTAX=SGML

Es soll die SGML-Syntax verwendet werden.

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:

{*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 einen positiven Offset

– numerischer Wert für einen negativen 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	0	12	5467,872	0,6785
#	0	12	5468	1
#.##	0,00	12,00	5467,87	0,68
###	000	012	5468	001

Wenn kein Format definiert wurde, gelten folgende Voreinstellungen:

pt # . #

mm # . ##

cm # . ###

in # . ####

% # . #

%% # . #

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="' +'">'
```

```
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-Konvertierungsregeln*, *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 PUBLIC ...">'
  BEG='<html >'
  END='</html >'
ENDIF
ENDE
```

Ergebnis:

```
<!DOCTYPE html PUBLIC ...">
<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 einen positiven Offset
 - numerischer Wert für einen negativen Offset.
- Bitte achten Sie auf die Eingabe des Doppelpunktes am Beginn des Offsets!
- Format* Definition des Ausgabezahlenformats. Bitte stets die Klammern setzen, z.B. (**# . ##**). Folgende Platzhalterzeichen können verwendet werden:
- #** Platzhalter für eine Dezimalzahl
 - .** Dezimalpunkt



Beispiele:

Beispiele	0	12	5467,872	0,6785
#	0	12	5468	1
#.##	0,00	12,00	5467,87	0,68
###	000	012	5468	001

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

pt #.#

mm #.##

cm #.###

in #.####

% #.#

%% #.#

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

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-Ausgabebezeichnung* 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-Konvertierungsregeln*, *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**

Fehlermeldungen

Der Converter gibt den Erfolg der Programmausführung als Error-level zurück:

- | | |
|----|---|
| 0 | Erfolg, keine Fehler aufgetreten |
| 1 | Parameter konnten nicht gelesen werden |
| 2 | Sonstige Fehler |
| 3 | Unbekannter Parameter gefunden |
| 4 | Kein Parameter angegeben |
| 5 | Hilfe wurde angezeigt, es wurde aber keine Konvertierung durchgeführt |
| 6 | Interpreter konnte nicht initialisiert werden |
| 7 | Kein Skript angegeben |
| 8 | Ausgabe im Kompatibilitätsmodus nicht erlaubt |
| 9 | Eingabemuster ungültig |
| 10 | Eingabedateiname ungültig |
| 11 | Ausgabedateiname ungültig |
| 12 | Ausgabedateinamenserweiterung ungültig |
| 13 | Datenfehler |

Weitere detaillierte Fehlermeldungen werden über die Standardausgabe angezeigt. Diese werden in der Konsole direkt angezeigt.

XML/SGML-Glossar

Übersicht über die wichtigsten XML-/SGML-Bestandteile in alphabetischer Reihenfolge.

Wichtiger Hinweis

Die Beschreibung ist jeweils nur auszugsweise!

ATTLIST

Deklaration bzw. Vereinbarung von Attributen zu einem Element.

Syntax XML

<!ATTLIST *bezeichner* {*attributname typ vorgabewert*}* **>**

bezeichner Ein Name des zugehörigen Elementes:

Elementname

attributname Name eines Attributes:

typ Typ des Attributes:

CDATA

ENTITY

ENTITIES

ID

IDREF

IDREFS

NMTOKEN

NMTOKENS

NOTATION

oder

ein Gruppe von Auswahlwerten.

vorgabewert Vorgabewert:

FIXED

IMPLIED

REQUIRED

oder
ein Vorgabewert (Zeichenkette).

Syntax SGML

<!ATTLIST *bezeichner* {*attributname typ vorgabewert*}^{*} **>**

bezeichner Ein Name oder eine Namensgruppe eines zugehörigen Elementes:

Elementname

(*Elementname* | *Elementname*|...)

attributname Name eines Attributes:

typ Typ des Attributes:

CDATA

ENTITY

ENTITIES

ID

IDREF

IDREFS

NAME

NAMES

NMTOKEN

NMTOKENS

NOTATION

NUMBER

NUMBERS

NUTOKEN

NUTOKENS

oder

ein 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:

<

>

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

Bezeichner XML-*Bezeichner*

Zeichendaten 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

Zeichendaten 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 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

[Einschluss](#)



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: ABCDEFGHIJKLMNOPQRSTUVWXYZ
 abcdefghijklmnopqrstuvwxyz
 :_

Alle folgenden Zeichen können aus dem erweiterten Zeichenbereich entnommen werden:

folgende Zeichen: ABCDEFGHIJKLMNOPQRSTUVWXYZ
 abcdefghijklmnopqrstuvwxyz
 0123456789
 : . - _

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: ABCDEFGHIJKLMNOPQRSTUVWXYZ
 abcdefghijklmnopqrstuvwxyz

Alle folgenden Zeichen können aus dem erweiterten Zeichenbereich entnommen werden:

folgende Zeichen: ABCDEFGHIJKLMNOPQRSTUVWXYZ



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 FF	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**)

Datenbereiche, die reine Textinformationen enthalten.

Es erfolgt kein Parsen, d.h. *Elemente* sind nicht erlaubt und *Entitäten* werden nicht ersetzt.

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 Unter-elementen 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.

Element-Deklaration XML:

<!ELEMENT *bezeichner inhaltsmodell* **>**

<i>bezeichner</i>	<i>Elementname</i>
<i>inhaltsmodell</i>	Definition des Inhaltsmodells: EMPTY leeres Element ANY beliebiger Inhalt erlaubt oder einer expliziten Inhaltsbeschreibung bestehen aus: #PCDATA Textdaten/Zeichendaten Elementnamen zulässige Elemente im Inhalt Opertoren ", " ", "(, ")", "?", "+"

Element-Deklaration SGML:

<!ELEMENT *bezeichner sm em inhaltsmodell excl incl* **>**

<i>bezeichner</i>	<i>Elementname</i>
<i>sm</i>	Kennzeichen für Starttag-Minimierung: o Starttag kann ausgelassen werden - Starttag muss vorhanden sein.
<i>em</i>	Kennzeichen für Endtag-Minimierung o Endtag kann ausgelassen werden - Endtag muss vorhanden sein.
<i>inhaltsmodell</i>	Definition des Inhaltsmodells: EMPTY leeres Element ANY beliebiger Inhalt erlaubt oder einer expliziten Inhaltsbeschreibung bestehen aus: #PCDATA Textdaten/Zeichendaten Elementnamen zulässige Elemente im Inhalt



Opertoren ", " "|", "(" , ")", "?", "+"

excl

Ausschluss:

- (Elementname oder Elementgruppe)

incl

Einschluss

+ (Elementname oder Elementgruppe)

Hinweis (XML Converter):

Die DTD und deren Bestandteile werden nicht ausgewertet.

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 Emptytag (`<bild/>`) in der XML-Syntax.

Der *Elementname* ist hierbei der Bezeichner des Start- End- oder Emptytags (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)

`</...>` Endemarke 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 wird 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:

NDATA {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***>**

Bezeichner Name der Entität (*Bezeichner*)

Text *Zeichenkette*, des definierten Textes.

<i>System-ID</i>	Zeichenkette , die einen Systembezeichner enthält
<i>Public-ID</i>	Zeichenkette , 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 `auml` 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;

& Beginn der Entität

Entitätenname **Bezeichner** der Entität

; 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 `ü` oder `ü` 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 DTD und deren Bestandteile werden nicht ausgewertet.

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

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*

<i>Public-ID</i>	eingeschränkte Zeichenkette , die einen öffentlichen Bezeichner enthält (siehe Zeichenkette , eingeschränkte)
<i>System-ID</i>	eingeschränkte Zeichenkette , die einen Systembezeichner (siehe Zeichenkette , eingeschränkte und SYSTEM) enthält

Syntax des Bezeichners

"*Norm*//*Besitzer*//*Klasse Spezifikation*//*Sprache*"

<i>Norm</i>	Ist dieser Bezeichner als Norm angemeldet:
	+ ISO angemeldet
	- nicht angemeldet
<i>Besitzer</i>	Kennzeichnung des Besitzers bzw. Firmenname
<i>Klasse</i>	Was wird definiert?
	DTD öffentlicher Bezeichner für eine DTD
<i>Spezifikation</i>	Der eigentliche öffentliche Bezeichner einschließlich der Versionskennzeichnung
<i>Sprache</i>	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, SHORTTAG} LINK {SIMPLE, IMPLICIT, EXPLICIT} OTHER {CONCUR, SUBDOC, FORMAL}
<i>info</i>	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:

<code><></code>	leerer Starttag Beginnt ein Element, welches durch Kontext und DTD eindeutig erkannt werden kann. Der Elementname kann in diesem Fall ausgelassen werden.
<code></></code>	leerer Endtag Schließt das offene Element entsprechend Kontext/DTD.
<code><tag1<tag2></code>	nichtgeschlossene (verkürzte) Tags. Ein Tag wird auch dadurch geschlossen (>), dass ein neuer Tag (<) begonnen wird.
<code><tag1 id=wert1></code>	Attributwerte, die nicht in Anführungszeichen bzw. Hochkommas gesetzt werden.
<code><tag1 wert1></code>	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.

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? ?>

version Version des XML-Dokumentes in Schreibweise eines Attributes:

version="zeichenkette"

Zur Zeit gültige Werte der XML-Version:

1.0 XML Version 1.0

Siehe auch **[XML]**.

encoding Kodierung des XML-Dokumentes in Schreibweise eines Attributes

encoding="zeichenkette"

Gültige Werte für die Kodierung sind z.B.:

ASCII ASCII-Kodierung

US-ASCII ASCII-Kodierung

UTF-8 UTF-8-Kodierung

...

Siehe auch **[XML]**.

standalone Definiert, ob das Dokument ohne externe Definitionen (DTD) gültig ist. Diese Angabe wird in Schreibweise eines Attributes angegeben:

encoding="zeichenkette"

Gültige Werte sind:



yes	Dokument ist "alleinstehend" gültig (keine DTD bzw. keine externe Definitionen erforderlich).
no	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

&#Funktion;

Funktion *Bezeichner* für das Zeichen, gültige vordefinierte Werte sind:

RE (13) Wagenrücklauf

RS (10) Zeilenvorschub

TAB (9) Tabulator

SPACE (32) Leerzeichen





Kompatibilitätsmodus

Hinweis

In neuen Skripten nicht verwenden!

Beschreibung

Dieser Modus ist zur Erhaltung der Kompatibilität mit Skripten, die für den Converter bis zur Version 1.23 entwickelt worden, erforderlich!

Übersicht über die Änderungen

Im Folgenden wird die Syntax des Kompatibilitätsmoduses mit

Alte Syntax

Hinweise zur neuen Syntax

Aufruf

Der Schalter /c wurde für den Kompatibilitätsmodus hinzugefügt. Skripte aus der Version 1.23 oder einer früheren Version können nur noch mit diesem Schalter gestartet werden.

Attributbedingung []

Die Vergleichsoperatoren "<>" und "=" werden in der neuen Syntax unabhängig der Groß- und Kleinschreibung ausgewertet:

<> ungleich (unabhängig der Groß- und Kleinschreibung)

= gleich (unabhängig der Groß- und Kleinschreibung)

Zusätzlich wurden folgende neue Operatoren eingeführt:

== identisch (mit Beachtung der Groß- und Kleinschreibung)

!= nicht identisch (mit Beachtung der Groß und Kleinschreibung)

Skript-Einstellungen

Die Einstellungen FILENAME, EXTENSION und OWNER wurden entfernt. Der Dateiname der Ausgabedatei oder die Dateierweiterung der Ausgabedateien werden in der neuen Syntax ausschließlich per Kommandozeilenparameter gesteuert (siehe Bedienung).

CONVERTELEMENTS	In der neuen Syntax wird als Vorgabe ONLYDEFINED verwendet, im Kompatibilitätsmodus ALL.
CONVERTENTITIES	In der neuen Syntax wird als Vorgabe ONLYDEFINED verwendet, im Kompatibilitätsmodus ALL.
ALTVALUE	VALUE(...,ASC)
ASCVALUE	VALUE(...,ALT)
BASE	In der alten Syntax wurde das Basiselement als <i>Zeichenkette</i> angegeben, in der neuen Syntax wird es "ohne Anführungszeichen" als in der Syntax eines <i>Bezeichners</i> angegeben.
COPY	Die Ersetzungstypen ALTFULL, ASCFULL und DROP entfallen in der neuen Syntax. Bei den Datentypen
COPYALL	COPY(...,ALL,...,{ <i>Trenntext</i> })
COPYALTALL	COPY(...,ALL,ALT,...,{ <i>Trenntext</i> })
COPYALTONE	COPY(...,ONE,ALT,...)
COPYASCALL	COPY(...,ALL,ASC,...,{ <i>Trenntext</i> })
COPYASCONE	COPY(...,ONE,ASC,...)
COPYONE	COPY(...,ONE,...)
COUNT	In der neuen Syntax wird das angegebene Element nicht nur im erweiterten Kontext gesucht, sondern es wird auch das eigene Element geprüft.
COUNTER	entfallen.
CUT	Die Ersetzungstypen ALTFULL, ASCFULL und DROP entfallen in der neuen Syntax.
EXTENSION	Die Dateierweiterung der Ausgabedateien wird in der neuen Syntax über einen Kommandozeilenparameter gesteuert (siehe <i>Bedienung</i>).



FILENAME	Der Dateiname der Ausgabedatei wird in der neuen Syntax über einen Kommandozeilenparameter gesteuert (siehe Bedienung).
OWNER	Diese Einstellung ist komplett entfallen.

Aufruf

Syntax

```
XMLcnv.exe eing skript {/s}? {/o}? {/l}? /c {/d}?
```

Einstellungen

In den *Skript-Einstellungen* tragen Sie erforderlichen Einstellungen, Zieldateierweiterung oder Zieldateiname, XML- oder SGML-Syntax, den PUBLIC und/oder SYSTEM-Identifikator der DTD usw. ein. Das Beispiel einer HTML-Konvertierung können kann wie folgt konfiguriert werden:

```
EXTENSION="html"
```

```
SYNTAX=XML
```

```
BASE="dokument"
```

```
OWNER="DEMO"
```

```
SYSTEM="dokument.dtd"
```

```
PUBLIC="-//Unser Beispiel//DTD Demo 1.0//DE"
```

In diesen Einstellungen wird festgelegt, 1) dass pro konvertierte Datei eine Zielfeile mit der Dateierweiterung "html" erzeugt wird, 2) dass die Ausgangsdaten XML-Daten sind, 3) wenn in den Instanzen kein DOCTYPE angegeben wird, dass die Instanzen nur gültig sind, wenn das Basiselement "dokument" lautet, 4) wenn in den Instanzen ein DOCTYPE mit einem SYSTEM-Identifikator angegeben wird, dass die Instanzen nur gültig sind, wenn der Eintrag "dokument.dtd" lautet, 5) wenn in den Instanzen ein PUBLIC-Identifikator angegeben wird, dass die Instanzen nur gültig sind, wenn der Eintrag "-//Unser Beispiel//DTD Demo 1.0//DE" lautet.

Hinweis

Im Kompatibilitätsmodus ist keine Definition einer Ausgabe erlaubt. Die Ausgabe muss im Skript definiert werden. Der Kompatibilitätsmodus wird entweder über Kommandozeilenschalter oder über die Dateierweiterung des Skriptes (ehemals: MTC) gesteuert.

Im Folgenden werden die Einstellungen/Befehle, die ausschließlich im Kompatibilitätsmodus gültig sind, aufgeführt:



- *Skript-Einstellungen*
- EXTENSION
- FILENAME
- OWNER

Attributbedingung []

Diese Kennung wird in Abfragen (*Attributbedingung*, *Bedingungsausdruck*) für die Abfrage von Attributwerten verwendet.

Hinweis

Diese Einstellungen gelten nur für den *Kompatibilitätsmodus*!

Syntax

[*Attributname* *Operator* *Wert*]

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

Operator Vergleichsoperator:

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

Wert *Zeichenkette*, für den *Attributwert*.

Diese Syntax wird innerhalb von *Attributbedingung* verwendet.

Beachten Sie, dass die Vergleichsoperatoren "=" und "<>" im Kompatibilitätsmodus stets die Groß- und Kleinschreibung berücksichtigt.

Die aktuelle Syntax (siehe []) verwendet hierfür die Operatoren "identisch (==)" und "nicht identisch (!=)" und für Vergleiche unabhängig der Groß- und Kleinschreibung "gleich (=)" und "ungleich (<>)".



Skript-Einstellungen

Parameter zur Konfiguration des *Skriptes*.

Soll eine Ausgabedatei je Eingabedatei oder eine einzige Ausgabedatei für alle Eingabedateien erzeugt werden?

- Siehe **EXTENSION** und **FILENAME**.

Sollen XML- oder SGML-Daten konvertiert werden?

- Siehe **SYNTAX**.

Sollen im Dokument alle Elemente und Entitäten vollständig konvertiert werden?

- Siehe **CONVERTELEMENTS** und **CONVERTENTITIES**.

Wer ist der Besitzer der Lizenz?

- Siehe **OWNER**.

Wie werden die passenden Daten zum *Skript* identifiziert?

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

Hinweis

Diese Einstellungen gelten nur für den **Kompatibilitätsmodus**!

Syntax

{**EXTENSION**|**FILENAME**}

{**SYNTAX**}?

{**CONVERTELEMENTS**}?

{**CONVERTENTITIES**}?

BASE

OWNER

SYSTEM

{**PUBLIC**}+

ALTVALUE

Ausgabe von beliebigen Attributwerten. Sucht das angegebene Attribut im aktuellen *Element* oder im angegebenen *Element* im erweiterten Kontext und gibt den Inhalt mit alternativer Entitäten-Ersetzung aus.

Syntax 1

ALTVALUE(*[Attributname]*)

Syntax 2

ALTVALUE(*Elementname*[*Attributname*])

<i>Attributname</i>	<i>Bezeichner</i> für den Namen eines <i>Attributes</i> .
<i>Elementname</i>	<i>Bezeichner</i> für ein <i>Element</i> . Dieses <i>Element</i> kann in einem beliebigen Kontext zu dem aktuellen <i>Element</i> stehen. Die Kontextzuordnung erfolgt entsprechend dem INX-Kommando.

Beispiel 1

Daten:

```
<kapitel nr="5&sol;4">
  <titel>Vorwort</titel>
</kapitel>
```

Skript:

```
ELEMENT kapitel
  ATTLIST [nr=""]
  BEG='<DIV><A TITEL="' + ALTVALUE ( [nr] ) + '">'
  END="</A></DIV>"
ENDE

ENTITY sol="&sol;" ALTERNATE="." ASCII="-"
```

Ergebnis:

```
<DIV><A TITEL="5.4">Vorwort</TITEL></DIV>
```

Mit der Attributabfrage (VALUE) wird der Wert des Attributes `nr` des Elementes `kapitel` abgefragt und als HTML-Element `a` mit dem

Attribut `titel` ausgegeben, wobei die Entität `sol` wie definiert als / ausgegeben wird.

Beispiel 2

Daten:

```
<kapitel nr="5&sol;4">
  <titel>Vorwort</titel>
</kapitel>
```

Skript:

```
ELEMENT titel
  BEG="<h1>"+ALTVALUE(kapitel[nr])+ " "
  END="</h1>"
ENDE

ENTITY sol="&sol;"; ALTERNATE="." ASCII="-"
```

Ergebnis:

```
<H1>5.4 Vorwort</H1>
```

Mit der Attributabfrage (VALUE) wird der Wert des Attributes `nr` des Elementes `kapitel` abgefragt und als Text der HTML-Überschrift `h1` ausgegeben, wobei die Entität `sol` wie definiert als / ausgegeben wird.

ASCVALUE

Ausgabe von beliebigen Attributwerten mit Entitätenersetzung für ASCII. Sucht das angegebene Attribut im aktuellen *Element* oder im angegebenen *Element* im erweiterten Kontext und gibt den Inhalt mit Entitäten-Ersetzung für ASCII aus.

Syntax 1

ASCVALUE([*Attributname*])

Syntax 2

ASCVALUE(*Elementname*[*Attributname*])

<i>Attributname</i>	<i>Bezeichner</i> für den Namen eines <i>Attributes</i> .
<i>Elementname</i>	<i>Bezeichner</i> für ein <i>Element</i> . Dieses <i>Element</i> kann in einem beliebigen Kontext zu dem aktuellen <i>Element</i> stehen. Die Kontextzuordnung erfolgt entsprechend dem INX-Kommando.

Beispiel 1

Daten:

```
<kapitel nr="5&sol;4">
  <titel>Vorwort</titel>
</kapitel>
```

Skript:

```
ELEMENT kapitel
  ATTLIST [nr=""]
  BEG='<DIV><A NAME="'+ASCVALUE([nr])+'" />'
  END="</DIV>"
ENDE

ENTITY sol="&sol;" ALTERNATE="." ASCII="-"
```

Ergebnis:

```
<DIV><A NAME="5-4" /></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, wobei die Entität `sol` wie definiert als - ausgegeben wird.

Beispiel 2

Daten:

```
<kapitel nr="5&sol;4">
  <titel>Vorwort</titel>
</kapitel>
```

Skript:

```
ELEMENT titel
  BEG="<h1>"+ASCVALUE(kapitel[nr])+ " "
  END="</h1>"
ENDE

ENTITY sol="&sol;" ALTERNATE="." ASCII="-"
```

Ergebnis:

```
<H1>5-4 Vorwort</H1>
```

Mit der Attributabfrage (VALUE) wird der Wert des Attributes `nr` des Elementes `kapitel` abgefragt und als Text der HTML-Überschrift `h1` ausgegeben, wobei die Entität `sol` wie definiert als - ausgegeben wird.

**BASE**

Im Kompatibilitätsmodus wird das Basiselement als Zeichenfolge und nicht als *Elementname* angegeben.

COPY

Im Kompatibilitätsmodus stehen folgende Ersetzungstypen im Befehl **COPY** zusätzlich zur Verfügung:

- ALTFULL
- ASCFULL
- DROP

Ein weiterer Unterschied besteht darin, dass die Ersetzungsregeln für das aufgerufene Element im Kompatibilitätsmodus immer ausgeführt werden!

Ersetzungstyp zusätzliche Werte im Kompatibilitätsmodus:

ALTFULL Wie FULL, aber Entitäten werden alternativ umgesetzt.

ASCFULL Wie FULL, aber Entitäten werden nach ASCII umgesetzt.

DROP Alle *Unterelemente* und Daten werden ausgelassen.

COPYALL

Sucht alle Vorkommen des angegebenen Elementes in der *Unterstruktur* des aktuellen Elementes und gibt die Daten zurück. Es werden keine Element-Regeln für die enthaltenen *Unterelemente* ausgeführt. Die Daten werden an der aktuellen Position eingefügt.

Werden mehrere Elemente gefunden, so werden diese mit dem angegebenen Trennungstext voneinander abgetrennt.

Syntax

COPYALL(*Elementname*,*Zeichenkette*)

Elementname *Bezeichner* für ein *Element*.

Zeichenkette Ein Text, der als Abtrennung jeweils zwischen die gefundenen Daten eingefügt wird.

Beispiel

Daten:

```
<dokument>
  <titel>Vorwort<index>B&auml;r</index></titel>
  <p>Das ist <b>wichtig<index>Wolf</index></b></p>
</dokument>
```

Skript:

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

Ergebnis:

```
<meta name="keywords" content="B&auml;r, Wolf">
```

In diesem Beispiel werden alle Stichwort-Elemente `index` gesucht und als HTML Meta-Element `keywords` ausgegeben.

COPYALTALL

Sucht alle Auftreten des angegebenen Elementes in der *Unterstruktur* des aktuellen Elementes und gibt die Daten zurück. Es werden keine Element-Regeln für die enthaltenen *Unterelemente* ausgeführt. Alle gefundenen Daten werden mit alternativer Entitäten-Ersetzung umgesetzt. Die Daten werden an der aktuellen Position eingefügt.

Syntax

ALTALL(*Elementname*,*Zeichenkette*)

Elementname *Bezeichner* für ein *Element*.

Zeichenkette Ein Text, der als Abtrennung jeweils zwischen die gefundenen Daten eingefügt wird.

Beispiel

Daten:

```
<dokument>
  <titel>Vorwort<index>B&auml;r</index></titel>
  <p>Das ist <b>wichtig<index>Wolf</index></b></p>
</dokument>
```

Skript:

```
ELEMENT dokument
  BEG='<meta name="keywords" content="'
  BEG=COPYASCALL(index," ")
  BEG="' ">'
ENDE
ENTITY Ouml="&auml" ALTERNATE="ae" ASCII="ä"
```

Ergebnis:

```
<meta name="keywords" content="Baer, Wolf">
```

In diesem Beispiel werden alle Stichwort-Elemente `index` gesucht und als HTML Meta-Element `keywords` ausgegeben. Die Entitäten-konvertierung erfolgt hierbei entsprechend `ALTERNATE`-Anweisung als `ae`.

COPYALTONE

Sucht das erste Auftreten des angegebenen Elementes in der *Unterstruktur* des aktuellen Elementes und gibt die Daten zurück. Es werden keine Element-Regeln für die enthaltenen *Unterelemente* ausgeführt. Alle gefundenen Daten werden mit alternativer Entitäten-Ersetzung umgesetzt. Die Daten werden an der aktuellen Position eingefügt.

Syntax

ALTONE(*Elementname*)

Elementname *Bezeichner* für ein *Element*.

Beispiel

Daten:

```
<dokument>
<titel>&Ouml;l ist wichtig</titel>
<titel>Vorwort</titel>
</dokument>
```

Skript:

```
ELEMENT document
  BEG='<meta name="titel" content="'
  BEG=COPYONE(index)
  BEG=' ">'
ENDE
ELEMENT titel
  DAT=DROP
ENDE
ENTITY Ouml="&Ouml;" ALTERNATE="Oe" ASCII="Ö"
```

Ergebnis:

```
<meta name="titel" content="Oel ist wichtig">
```

In diesem Beispiel wird der erste Titel (Element `titel`) im Dokument gesucht und in der HTML-Ausgabe mit alternativer Ersetzung gemäß der `ALTERNATE`-Definition ausgegeben:



COPYASCALL

Sucht alle Vorkommen des angegebenen Elementes in der *Unterstruktur* des aktuellen Elementes und gibt die Daten zurück. Es werden keine Element-Regeln für die enthaltenen *Unterelemente* ausgeführt.

Alle gefundenen Daten werden entsprechend der definierten ASCII-Entitäten-Ersetzung umgesetzt. Die Daten werden an der aktuellen Position eingefügt.

Werden mehrere Elemente gefunden, so werden diese mit dem angegebenen Trennungstext voneinander abgetrennt.

Syntax

ASCALL(*Elementname*,*Zeichenkette*)

Elementname *Bezeichner* für ein *Element*.

Zeichenkette Ein Text, der als Abtrennung jeweils zwischen die gefundenen Daten eingefügt wird.

Beispiel

Daten:

```
<dokument>
  <titel>Vorwort<index>B&auml;r</index></titel>
  <p>Das ist <b>wichtig<index>Wolf</index></b></p>
</dokument>
```

Skript:

```
ELEMENT dokument
  BEG='<meta name="keywords" content="'
  BEG=COPYASCALL(index," ")
  BEG="'>'
ENDE
ENTITY Ouml="&auml" ALTERNATE="ae" ASCII="ä"
```

Ergebnis:

```
<meta name="keywords" content="Bär, Wolf">
```

In diesem Beispiel werden alle Stichwort-Elemente `index` gesucht und als HTML META-Element `keywords` ausgegeben. Die Entitätä-



tenkonvertierung erfolgt hierbei entsprechend ASCII-Anweisung als
ä.

COPYASCONES

Sucht das erste Auftreten des angegebenen Elementes in der *Unterstruktur* des aktuellen Elementes und gibt die Daten zurück. Es werden keine Element-Regeln für die enthaltenen *Unterelemente* ausgeführt.

Alle gefundenen Daten werden entsprechend der definierten ASCII-Entitäten-Ersetzung umgesetzt. Die Daten werden an der aktuellen Position eingefügt.

Syntax

ASCONE(*Elementname*)

Elementname *Bezeichner* für ein *Element*.

Beispiel

Daten:

```
<dokument>
<titel>&Ouml;l ist wichtig</titel>
<titel>Vorwort</titel>
</dokument>
```

Skript:

```
ELEMENT document
  BEG='<meta name="titel" content="'
  BEG=COPYONE(index)
  BEG=' ">'
ENDE
ELEMENT titel
  DAT=DROP
ENDE
ENTITY Ouml="&Ouml;" ALTERNATE="Oe" ASCII="Ö"
```

Ergebnis:

```
<meta name="titel" content="Öl ist wichtig">
```

In diesem Beispiel wird der erste Titel (Element `titel`) im Dokument gesucht und in der HTML-Ausgabe mit ASCII-Ersetzung ausgegeben.



COPYONE

Sucht das erste Auftreten des angegebenen Elementes in der *Unterstruktur* des aktuellen Elementes und gibt die Daten zurück. Es werden keine Element-Regeln für die enthaltenen *Unterelemente* ausgeführt. Die Daten werden an der aktuellen Position eingefügt.

Syntax

COPYONE(*Elementname*)

Elementname *Bezeichner* für ein *Element*.

Beispiel

Daten:

```
<dokument>
<titel>&Ouml;l ist wichtig</titel>
<titel>Vorwort</titel>
</dokument>
```

Skript:

```
ELEMENT dokument
  BEG='<meta name="titel" content="'
  BEG=COPYONE(index)
  BEG=' ">'
ENDE
ELEMENT titel
  DAT=DROP
ENDE
ENTITY Ouml="&Ouml" ALTERNATE="Oe" ASCII="Ö"
```

Ergebnis:

```
<meta name="titel" content="&Ouml;l ist wichtig">
```

In diesem Beispiel wird der erste Titel (Element `titel`) im Dokument gesucht und in der HTML-Ausgabe komplett mit Entitäten ausgegeben.



COUNT

Im Kompatibilitätsmodus wird das angegebene Element ausschließlich im erweiterten Kontext gesucht.

COUNTER

Zählt das Vorkommen aller Elemente auf einer Ebene, entweder vom aktuellen *Element* oder vom angegebenen *Element* aus. Bei Angabe eines Elementes wird dieses im erweiterten Kontext gesucht und von der gefundenen Position aus werden alle Elemente dieser Ebene gezählt. Der Ergebniswert wird an der aktuellen Position eingefügt.

Hinweis

Es werden die Elemente der gefundenen Ebene gezählt. In der gefundenen Ebene werden die Elemente unabhängig vom Namen gezählt. Der angegebene Elementname dient lediglich dem Auffinden der Ebene und der Position, bis zu der gezählt werden soll.

Im Unterscheid zu `COUNT` werden auch unterschiedliche Elemente der Ebene gezählt. `COUNT` zählt nur das angegebene Element in der Ebene.

Syntax 1

COUNTER()

Syntax 2

COUNTER(*Elementname*)

Elementname *Bezeichner* für ein *Element*.

Beispiel

Daten:

```
<dokument>
  <kapitel>
    <titel>Vorwort</titel>
  </kapitel>
```

```
<kapitel>
  <titel>Anleitung</titel>
</kapitel>
<kapitel>
  <titel>Zusammenfassung</titel>
</kapitel>
<anhang>
  <titel>Hinweise</titel>
</anhang>
</dokument>
```

Skript:

```
ELEMENT titel
IF IN kapitel
  BEG="<h1>"+COUNTER(kapitel)+" . "+
  END=" (" +COUNT(kapitel)+") </h1>"
ELSEIF IN anhang
  BEG="<h1>"+COUNTER(anhang)+" . "
  END=" (" +COUNT(anhang)+") </h1>"
ENDIF
ENDE
```

Ergebnis:

```
<h1>1. Vorwort (1)</h1>
<h1>2. Anleitung (2)</h1>
<h1>3. Zusammenfassung (3)</h1>
<h1>4. Hinweise (1)</h1>
```

In diesem Beispiel werden die Überschriften der einzelnen Kapitel durchnummeriert. Zum verdeutlichen des Unterschiedes zum COUNT-Befehls wurden am Ende der Überschriften die Zählwerte von COUNT in Klammern zusätzlich angegeben.

CUT

Im Kompatibilitätsmodus stehen folgende Ersetzungstypen im Befehl **CUT** zusätzlich zur Verfügung:

- ALTFULL
- ASCFULL
- DROP

Ein weiterer Unterschied besteht darin, dass die Ersetzungsregeln für das aufgerufene Element im Kompatibilitätsmodus immer ausgeführt werden!

Ersetzungstyp zusätzliche Werte im Kompatibilitätsmodus:

ALTFULL Wie FULL, aber Entitäten werden alternativ umgesetzt.

ASCFULL Wie FULL, aber Entitäten werden nach ASCII umgesetzt.

DROP Alle *Unterelemente* und Daten werden ausgelassen.

EXTENSION

Definiert die Dateierweiterung (-extension) der zu erzeugenden Dateien. Die Extension der Zieldatei sollte sich von der Quelldatei unterscheiden. Sind die Bezeichnungen von Quell- und Zieldatei identisch, wird die Quelldatei überschrieben!

Hinweis

Diese Einstellungen gelten nur für den Kompatibilitätsmodus!

Syntax

EXTENSION = *Text*

Text *Zeichenkette*, der durch Hochkommas ("...") oder '...' angegeben werden muss.



Dieser Text definiert die zu verwendende Dateinamenserweiterung für die Ausgabedatei. Es dürfen nur Zeichen verwendet werden, die vom verwendeten Betriebssystem für Dateibezeichnungen (bzw. Erweiterungen) zugelassen sind.

Alternativ kann mit dem Befehl `FILENAME` eine Datei als Zieldatei festgelegt werden. Dabei ist zu beachten, dass bei der Bearbeitung von mehreren Ausgangsdateien nur eine Zieldatei erzeugt wird.

Beispiel

```
EXTENSION = ".htm"
```

Für jede zu konvertierende Datei wird eine Ausgabedatei mit der Dateinamenserweiterung `htm` erzeugt.



FILENAME

Definiert den Dateinamen der zu erzeugenden Datei.

Bei der Bearbeitung von mehreren Ausgangsdateien wird hierbei nur eine Zielformat erzeugt. Die Extension der Zielformat sollte sich von der Quelldatei unterscheiden. Sind die Bezeichnungen von Quell- und Zielformat identisch, wird die Quelldatei überschrieben!

Die Ausgabeformat wird, wenn kein Verzeichnis angegeben wurde, im Ordner der Eingabedateien ausgegeben.

Hinweis

Diese Einstellungen gelten nur für den Kompatibilitätsmodus!

Syntax

FILENAME = *Text*

Text Zeichenkette, der durch Hochkommas ("..." oder '...') angegeben werden muss.

Dieser Text definiert den Dateinamen der Ausgabeformat. Es dürfen nur Zeichen verwendet werden, die vom verwendeten Betriebssystem für Dateibezeichnungen zugelassen sind.

Alternativ kann mit dem Befehl EXTENSION eine Dateierweiterung für die Zielformate festgelegt werden. Dabei wird für jede bearbeitete Datei eine Zielformat mit der angegebenen Extension erzeugt.

Beispiel

```
FILENAME = "kapitel.htm"
```

Für alle zu konvertierenden Dateien wird in eine gemeinsame Ausgabeformat kapitel.htm erzeugt, in der das Konvertierungsergebnis in Reihenfolge der Bearbeitung gespeichert wird.



OWNER

Definiert den Lizenznehmer. Über diesen Eintrag wird der Bezug von *Skript* und Programm hergestellt. Der Eigner-Eintrag muss mit dem Lizenzeintrag des Programms übereinstimmen.

Eine Programmausführung ist nur bei Übereinstimmung des Lizenzeintrages im *Skript* möglich.

Hinweis

Diese Einstellungen gelten nur für den Kompatibilitätsmodus!

Syntax

OWNER = *Text*

Text Ein Textbereich, der durch Hochkommas ("..." oder '...') angegeben werden muss.

Dieser Text enthält die Bezeichnung des Eigentümers.

Beispiel

```
OWNER = "VIELDRUCK GmbH"
```

Es werden nur Skripte verarbeitet, deren Lizenzkennung mit der produktinternen Lizenzkennung übereinstimmen.



Editoren

CodePad Konfiguration

Konfigurationsdateien

Editor\CodePad\XML Converter.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 der Regel: "C:\Programme") im Unterordner ("CodePad\Syntax Definitions").
- Starten Sie CodePad. Fertig.

Siehe

[CodePad]



Notepad++ Konfiguration

Konfigurationsdateien

Editor\NotepadPP\userDefineLang.xml

Hinweis

Diese Konfiguration wurde für **Notepad++ Version 5.2** (deutsche Version) erstellt.

Installation

- Gehen Sie mit dem Explorer in den Benutzerordner des Programms Notepad++. Dieser Ordner befindet sich im Benutzerverzeichnis ("C:\Dokumente und Einstellungen\{Benutzername}\Anwendungsdaten\Notepad++").
- Befindet sich in diesem Ordner noch keine Datei mit dem Namen "userDefineLang.xml", dann können Sie die mitgelieferte Konfigurationsdatei "userDefineLang.xml" direkt in diesen Ordner kopieren.
- Befindet sich in diesem Ordner bereits eine Datei mit dem Namen "userDefineLang.xml", dann müssen Sie diese mit einem Texteditor zur Bearbeitung öffnen. Öffnen Sie weiterhin die mitgelieferte Konfigurationsdatei ("userDefineLang.xml") mit einem Texteditor und kopieren Sie das Element "UserLang" komplett vom Starttag "<UserLang ...>" bis einschließlich dem Endtag "</UserLang>" in die Zwischenablage. Schließen Sie die mitgelieferte Konfigurationsdatei. Fügen Sie in die Konfigurationsdatei des Programms vor dem Endtag "</NotepadPlus>" den Inhalt der Zwischenablage ein.
- Starten Sie Notepad++. Fertig.

Siehe

[Notepad++]



PSPad Konfiguration

Konfigurationsdateien

Editor\PsPad\XML Converter.INI

Editor\PsPad\XML Converter.DEF

Hinweis

Diese Konfiguration wurde für den **PSPad Editor Version 4.5.3** (deutsche Version) erstellt.

Installation

- Kopieren Sie die Syntax-Konfigurationsdatei (.INI) in den Syntaxordner des PSPad. Dieser befindet sich im Programmverzeichnis (in der Regel: "C:\Programme") im Unterordner ("PSPad Editor\Syntax").
- Kopieren Sie die Kontext-Konfigurationsdatei (.DEF) in den Kontextordner des PSPad. Dieser befindet sich im Programmverzeichnis (in der Regel: "C:\Programme") im Unterordner ("PSPad Editor\Context").
- Starten Sie den PSPad Editor
- 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 "XML Converter" aus (Anklicken). In der linken Spalte wird jetzt automatisch der Eintrag "<not assigned>" auf "XML Converter" 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" und ".mtc" 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". Wiederholen Sie diesen Vorgang für die Endung ".mtc". 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.

Siehe

[PSPad]



TextPad Konfiguration

Konfigurationsdateien

Editor\TextPad\XML Converter.syn

Hinweis

Diese Konfiguration wurde für den **TextPad Version 5.2.0** (deutsche Version) erstellt.

Installation

- Kopieren Sie die Syntax-Konfigurationsdatei (.syn) in den Systemordner des Programms TextPad. Dieser befindet sich im Programmverzeichnis (in der Regel: "C:\Programme") im Unterordner ("TextPad 5\System").
- Starten Sie TextPad.
- Gehen Sie in das Menü "Konfiguration" - "Neue Dokumentklasse..." und tragen Sie hier "XML Converter". Gehen Sie auf "Weiter" und tragen Sie die Dateitypen (Elemente) wie folgt ein "*.xb, *.mtc". Gehen Sie auf "Weiter" und aktivieren Sie "Syntax-Hervorhebungen einschalten" und wählen Sie in der Auswahlliste "Syntaxdefinitionsdatei" den Eintrag "XML Converter.syn" aus. Folgen Sie den weiteren Anweisungen.
- Beenden Sie das Programm und starten Sie es erneut. Fertig.

Siehe

[TextPad]



UltraEdit Konfiguration

Konfigurationsdateien

Editor\UltraEdit\XML Converter.uew

Hinweis

Diese Konfiguration wurde für den **UltraEdit Version 14.20.1** (deutsche Version) erstellt.

Installation

- Kopieren Sie die Syntax-Konfigurationsdatei (XML Converter.uew) in den Benutzerordner des Programms UltraEdit. Dieser befindet sich im Benutzerverzeichnis ("C:\Dokumente und Einstellungen\{Benutzername}\Anwendungsdaten\IDMComp\UltraEdit").
- Gehen Sie mit dem Explorer in den Benutzerordner des Programms UltraEdit. Sichern Sie die Datei "WORDFILE.UEW" als Kopie (z.B. "WORDFILE_BACKUP.UEW").
- Öffnen Sie die Syntax-Konfigurationsdatei (XML Converter.uew) mit einem Texteditor und kopieren Sie den kompletten Inhalt in die Zwischenablage.
- Öffnen Sie danach die Datei "WORDFILE.UEW" mit einem Texteditor und fügen Sie den Inhalt der Zwischenablage an das Ende der Datei an. Speichern Sie diese Datei.
- Starten Sie UltraEdit. Fertig.

Siehe

[UltraEdit]

Systemanforderungen

Betriebssysteme

- Windows 95
- Windows 98
- Windows NT 4
- Windows 2000
- Windows XP

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 Größe der *Bezeichner* (*Attributname*, *Elementname* und Entitätenname) ist auf maximal 255 Zeichen begrenzt.

Die maximale Größe einer *Zeichenkette* (*Attributwerte*) ist auf 4096 Zeichen (4 KByte) begrenzt.

Ausführungsanweisungen dürfen in den Daten enthalten sein. Ausführungsanweisungen werden aber ignoriert. Die maximale Größe der Zeichendaten innerhalb einer *Ausführungsanweisung* ist auf 16384 (16 KByte) begrenzt.

XML

Alle Zeichen eines Dokumentes müssen entsprechend **UTF-8** oder **ASCII** (auch US-ASCII bzw. ANSI) 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:;<=>?@ ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_` 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 = 4096 (4 KByte)
- PILEN = 16384 (16 KByte)
- **NAMECASE** GENERAL=**YES** ENTITY=**NO**

Deklarationen

- #DEFAULT nicht zulässig

DTD

- Elementinhalte dürfen 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 ...>**
- Markierte Bereiche **<![...]>**
- mit der Ausnahme der **ENTITY NDATA** 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.



Entwicklungsgeschichte

(Achtung: Die Auflistung wird in zeitlich umgekehrter Reihenfolge wiedergegeben. Somit startet die Auflistung mit der neuesten Version!)

Version Datum - Beschreibung

XML Converter

- kostenfreie Software -

1.27.2 07.03.2009

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.
- PILEN (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 08.02.2009

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 Erstzungstypen/Löschtypen ungleich FULL definiert sind.
- COPY und CUT erhalten als letzten Optionalen Pa-

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. {, 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.

Korrekturen:

- Probleme mit dem ENDE-Befehl direkt am Dateiende behoben.
- Fehlermeldung in [], VALUE, IMPLIED und TOKEN bei nichtvorhandenen Elementen oder bei leeren oder nichtvorhandenen Attributwerten beseitigt.
- Formatierung der TOKEN-Werte korrigiert.
- XML-Verarbeitungsanweisungen (Processing Instructions, PI) wurden normgerecht mit einem Anwendungsziel (PI Target) erweitert.
- SGML-Verarbeitungsanweisungen (Processing Instructions, PI) wurden normgerecht auf beliebigen Text erweitert.
- Editor Konfiguration korrigiert.

1.26.4 18.01.2009

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 14.01.2009

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:

- NOTEMPTY hinzugefügt.

1.26.1 08.01.2009

Probleme mit leeren erweiterten Vereinbarungen im DOCTEYPE der Dokumente beseitigt.

1.25.1 31.01.2008 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 nicht-definierten 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. Das Format kann mit "(#.###)" z.B. auf dreistellig definiert werden.
- VALUE und IMPLIED jetzt alternativ zum Format auch mit Ersetzungstyp ASC oder ALT einstellbar.
- COUNT zählt jetzt auch römisch mit dem Parameter ROM (i, ii, iii, iv, ...) und UPROM (I, II, III, IV, ...).
- Die Beschränkung der COUNT-Ergebnisse wird erweitert auf 32 Zeichen. Bei Überschreitung dieser Größe wird das Ergebnis abgeschnitten, eine Fehlermeldung ausgegeben, die Konvertierung aber normal fortgesetzt.
- COUNT zählt jetzt auch die Anzahl der Elemente im gesamten Dokument.
- Die Ersetzungstypen bei COPY und CUT wurden vereinheitlicht, die Ersetzungstypen ALTFULL, ALT-ASC und DROP wurden entfernt. Die Elementregeln des eigenen Elementes werden nur noch beim Ersetzungstyp FULL ausgeführt.
- Alle Kontextabfragen jetzt einheitlich mit Attributbedingungen möglich.
- Neuer Befehl NOTSUB als Negation von SUB.

XML Converter

Produktname: „mediaTEXT XML Converter“

1.24 16.01.2003

beta

Testversion mit optimierten Kontextabfragen.

1.23 16.01.2003

- 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 30.08.2002

- 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 08.08.2002

- 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 01.08.2002

- Fehler bei der Entitätenkonvertierung innerhalb von Attributen beseitigt.

1.17 27.06.2002

- 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, COPY-ASCONE, etc. funktionieren wieder wie original definiert.

1.16 06.06.2002

- Rekursionsschutz für die Befehle COPY, CUT und DEL hinzugefügt.
- Skriptbefehle entsprechend der Groß-/Kleinschreibung (XML) für Benennungen der Elemente und Attribute umgestellt.

1.15 31.05.2002

- 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 27.05.2002

- DAT=DROP in allen Kontexten erlaubt.
- CUT mit allen Ersetzungstypen wie COPY definiert.
- DEL hinzugefügt (Typ FULL oder DATA).

1.13 16.05.2002

- 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 02.05.2002

- Problem mit Groß-/Kleinschreibung der Attributnamen im Skript beseitigt. Elemente und Attribute können XML-konform in der definierten Groß-/Kleinschreibung angegeben werden.

- XML als Vorgabe definiert.
 - Probleme mit CUT beseitigt.
- 1.11 26.04.2002
- Probleme mit temporären Dateien beseitigt. MS Visual C++ Bug (Article ID: Q51326).
- 1.10 (b) 18.04.2002
- interne Programmänderung, Modul für SGML-Deklaration eingefügt. Steuerung von XML/SGML jetzt über dieses Modul.
- 1.10 11.04.2002
- Attributausgabe im Konvertiermodus für undefinierte Elemente eingefügt.
 - XML- und SGML-Empty-Elemente werden auch im Konvertiermodus korrekt bestimmt.
- 1.09 17.03.2002
- 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!
- 1.08 22.02.2002
- BEG/END/DAT/ERR-Regelsätze korrigiert. Funktionsweise geprüft.
 - COPY/Cutt-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 bear-

beitet! Es werden nur die reinen Daten konvertiert bzw. ausgelassen

- COPY/Cutt-Funktionsweise korrigiert. Die angegebenen 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
COPYALSTONE:	ALSTONE
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!

SGML/XML Converter

Produktname: „mediaTEXT SGML Konverter“

1.07 15.02.2002

- COPY/CUT/SUB jetzt parametergesteuert über Kon-

textnamen. 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

1.06 07.02.2002

- 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 Parameter-

wert 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.

1.05 31.01.2002

- Bedingungstypen erweitert: BEFORE und NOTBEFORE, AFTER und NOTAFTER, FIRST und NOTFIRST, LAST und NOTLAST, IN und NOTIN, INX und NOTONX, TOPLEVEL und NOTTOPLEVEL, sowie FIND und NOTFIND.
- Beliebige Verschachtelungen von Kontextbefehlen jetzt möglich (IN, NOTIN, INX, NOTINX, BEFORE, NOTGEFORE, AFTER, NOTAFTER, FIRST, NOTFIRST, LAST und NOTLAST).

1.04 24.01.2002

- 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)
```

1.03B 17.01.2002

- 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.

1.03A 10.01.2002

- 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.

1.02E 19.07.2001

Problem der COUNT-Funktion der Version 1.02D behoben.

1.02D 05.07.2001

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.

1.02C 02.05.2000

- Fehlermeldung bei nicht angegeben Skriptnamen hinzugefügt.
- Fehlermeldung bei abgelaufener Lizenz hinzugefügt.

- XML-Namensbereich mit ":" hinzugefügt.

1.02B 26.04.2000

- 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.

1.02A 19.04.2000

- 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

SGML Converter

1.01A 13.04.2000

- Debugmodus mit Kommandozeilenschalter "/d" hin-

zugefügt.

- Debugmeldungen im Skript werden mit Zeilen- und Zeichennummer angegeben.
- Problem mit Abfragen der Vorgabewerte von Attributen in übergeordneten Elementen behoben

1.00J 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.

1.00I 09.11.1999

- Problem bei der Bearbeitung von COPY/CUT mit erweiterten Kontextbedingungen beseitigt.

1.00H 02.07.1999

- TOKEN: Fehler bei Summenberechnung ohne Angabe eines Korrekturfaktors oder einer neuen Einheit beseitigt!

1.00G 20.06.1999

- 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.

1.00F 12.06.1999

- 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.

1.00E 03.06.1999

- Erweiterte Fehlermeldung für COUNT
- Geänderte Struktur für Start- und Endtags mit erweiterten Fehlermeldungen
- Erweiterte Fehlermeldungen bei VALUE, ASC-VALUE 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.

1.00D 27.05.1999

- ALT- und ASC-VALUE hinzugefügt
- FILENAME hinzugefügt zur Erzeugung einer einzigen Ausgabedatei bei mehreren Eingabedateien.
- COPY und CUT kann per Parametersteuerung jetzt wahlweise alle (ALL) oder nur den ersten Treffer (ONE) umsetzen

1.00A 18.05.1999

1.00B

1.00C

- ASCII-Ersetzung hinzugefügt.
- TOKEN erweitert (%).
- ASCALL und ASCONE als zusätzliche ASCII-Funktionen hinzugefügt.
- EMPTY-Bedingung hinzugefügt.

1.00 13.04.1999

Erste ausgelieferte kommerzielle Version.

Produktname: „mediaTEXT SGML Konverter“

- 28.01.1999
Start 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. www.iana.org/assignments/character-sets
[SGML]	SGML Syntax Summary Table of Contents xml.coverpages.org/sgmlsyn/contents.htm
[UNICODE]	The Unicode Standard www.unicode.org/standard/standard.html
[Web SGML]	ISO 8879 TC2, Annex K Web SGML Adaptations www.y12.doe.gov/sgml/wg8/document/1955.htm
[XML]	Extensible Markup Language (XML) 1.0 (Fourth Edition) W3C Recommendation 16 August 2006, edited in place 29 September 2006 www.w3.org/TR/xml/

Editoren

[CodePad]	CodePad code and hex editor Version 4.1 vom 16.12.2006 shicola.wz.cz/codepad
[NotePad++]	NotePad++ Version 5.2 vom 08.02.2009 notepad-plus.sourceforge.net
[PSPad]	PSPad Freeware Editor Version 4.5.3 vom 25.11.2007 www.pspad.com
[TextPad]	TextPad Text Editor for Windows Version 5.2.0 vom 10.03.2008 www.textpad.com
[UltraEdit]	UltraEdit Text Editor Version 14.20.1 vom 26.11.2008 www.ultraedit.com