

Von XML nach PDF über T_EX– Tutorium

Günter Partosch, Oktober 2002

1 Übersicht

Übersicht

- Übersicht über die Verfahren
- Detaillierter Blick auf zwei Verfahren
 - XMLTEX + PassiveT_EX
 - ConT_EXt
- Internet-Adressen zum Thema

Die Folien für dieses Tutorium werden in regelmäßigen Abständen überarbeitet.
Die jeweils neueste Version finden Sie unter

<http://www.uni-giessen.de/partosch/xml/XML+TeX/>

Close

Von XML nach PDF über T_EX – Übersicht über die Verfahren

Günter Partosch, Oktober 2002

XML und PDF:

zwei prominente und weit verbreitete Standards in der
Dokumentenverarbeitung

Zu zeigen:

A. Wie passen XML und PDF zueinander?

⇒ Anmerkungen zu „XML“ und „PDF“, sowie die Betrachtungen
zu den Arbeitsabläufen („XML“ und „PDF“).

B. „Warum sollte hier T_EX eine Rolle spielen?“

C. Und wie sollte T_EX hier eine Rolle spielen?

⇒ Betrachtung der „Arbeitsabläufe“.

XML

PDF

Arbeitsabläufe

Resümee

Nicht ...

Close

Literatur zum Thema:

- Hans Hagen: **XML in ConT_EXt**
- Berend de Boer (EuroT_EX2000): **From database to presentation via XML, XSLT and ConT_EXt**
- Simon Pepping (EuroT_EX2000): **From XML to TeX: an overview of available methods**
- Eitan M. Gurari (TUG99): **LaTeX to XML/MathML**
- Robin Cover: **The XML Cover Pages. SGML/XML and (La)TeX**

XML

PDF

Arbeitsabläufe

Resümee

Nicht ...

Close

Warum sollte hier T_EX eine Rolle spielen?

- Dokumente, die hohen typographischen Anforderungen genügen, werden immer benötigt. – Und T_EX erfüllt diese Qualitätsansprüche.
- T_EX kann qualitativ hochwertige PDF-Dateien erzeugen.
- T_EX ist fehlerfrei und läuft nahezu stabil.
- T_EX läuft auf Wunsch batch-artig ab und kann hervorragend in Tool-Ketten eingesetzt werden.
- T_EX ist schnell und kann auch sehr große Dokumente aufbereiten.
- Die Darstellung mathematischer Formeln in T_EX ist unübertroffen gut.
- Trennungen durch T_EX sind gut.
- **Wir kennen T_EX (plainT_EX, L^AT_EX, ConT_EXt, usw.)!**

XML

PDF

Arbeitsabläufe

Resümee

Nicht ...

Close

1 XML

Einige Anmerkungen zu XML:

- *eXtensible Markup Language* (XML) ist ein Regelwerk zum Erstellen und Benutzen eigener Markup-Sprachen.
- Entwicklung und Pflege durch das W3-Konsortium: **Extensible Markup Language (XML) 1.0 (Second Edition)**
- dort auch umfangreiche Entwicklungen im Umfeld: **XML Information Set**
- Einsatzgebiete:
 - plattformunabhängige Archivierung von Dokumenten
 - Austausch von Dokumenteninhalten zwischen verschiedenen Plattformen und Institutionen
- Es gibt strenge Regeln für die Anwendung: XML-Dokument ist *wohlgeformt*.
- Und noch strenger: Dokument ist *gültig* (gehört einer DTD).

XML

PDF

Arbeitsabläufe

Resümee

Nicht ...

Close

- XML selbst macht *keine Aussagen über Darstellung/Präsentation* eines Dokuments.
- Es gibt aber *normierte Style-Konzepte* für die Darstellung/Präsentation:
 - DSSSL (*Document Style Semantics and Specification Language*); z.B. bei James Clark: **ISO/IEC 10179:1996. Document Style Semantics and Specification Language (DSSSL)**
 - **CSS1** + **CSS2** (*Cascading Style Sheets*), **CSS3** (in Arbeit)
 - XSL (*Extensible Stylesheet Language*):
 - ★ XSLT (*Extensible Stylesheet Language – Transformations*): **XSL Transformations (XSLT) Version 1.0**
 - ★ XSL FO (*Extensible Stylesheet Language – Formatting Objects*): **Extensible Stylesheet Language (XSL) Version 1.0**

XML

PDF

Arbeitsabläufe

Resümee

Nicht ...

Close

Einige prominente XML-Dokumenttypen:

- [DocBook](#)
- [Text Encoding Initiative](#)
- [XHTML 1.0: The Extensible HyperText Markup Language – A Reformulation of HTML 4 in XML 1.0](#)
- [MathML \(Mathematical Markup Language\): Mathematical Markup Language \(MathML\) 1.01 Specification](#) und [W3C's Math Home Page](#);

Ausführliche Dokumentation bei Hans Hagen:

- [MathML](#) (Handbuch),
- [MATHML](#) (Übersicht) und
- [MathML in ConT_EXt](#) (Beispiele)

XML

PDF

Arbeitsabläufe

Resümee

Nicht ...

Close

2 PDF

Einige Anmerkungen zu PDF:

- *Portable Document Format* (PDF) ist ein portables Darstellungsformat für Dokumente.
- Es ist weltweiter, geräte- und herstellerunabhängiger Standard.
- PDF wurde entwickelt von der Fa. Adobe; aktuelle Version ist 1.4 (Acrobat 5).
- Es basiert auf PostScript; verzichtet aber auf die Programmierbarkeit und enthält dafür ausgefeilte Möglichkeiten für Hypertextstrukturen.
- PDF beinhaltet einen gut durchdachten Font-Ersatz-Mechanismus (multiple master fonts).
- PDF ist ein Dokument-Endformat, d.h. nur noch wenige, spezielle Änderungen am Dokument sind möglich.
- PDF-Dokumente lassen sich gegen Drucken, Verändern, Entnehmen schützen.

XML

PDF

Arbeitsabläufe

Resümee

Nicht ...

Close

- PDF-Dokumente können überall mit geeignetem Viewer betrachtet bzw. auf Drucker ausgedruckt werden.
- PDF-Dateien sind deutlich kleiner als entsprechende PostScript-Dateien.

XML

PDF

Arbeitsabläufe

Resümee

Nicht ...

Close

3 Arbeitsabläufe

Unsere Ausgangssituation (drei Beispieldokumente):

kapitel0.xml:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!-- kapitel0.xml -->

<einkapitel>
  <titel>Titel des ersten Kapitels</titel>
  <rumpf>
    <absatz>Ein bisschen Text ...</absatz>
    <absatz>Noch ein bisschen Text ... </absatz>
  </rumpf>
</einkapitel>
```

XML

PDF

Arbeitsabläufe

Resümee

Nicht ...

Close

kapitel2.xml:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<?xml-stylesheet href="kapitel2-tex1.xsl" type="text/xsl" ?>
<!DOCTYPE mehrerekapitel SYSTEM "kapitel2.dtd" >

<!-- kapitel2.xml -->
<mehrerekapitel>
<einkapitel>
  <titel>Titel des ersten Kapitels</titel>
  <rumpf>
    <absatz>Ein bisschen Text ...</absatz>
    <absatz>Noch ein bisschen Text ... </absatz>
  </rumpf>
</einkapitel>

<einkapitel>
  <titel>Titel des zweiten Kapitels</titel>
  <rumpf>
    <absatz>Ein bisschen anderer Text ...</absatz>
    <absatz>Noch ein bisschen anderer Text ... </absatz>
  </rumpf>
```

XML

PDF

Arbeitsabläufe

Resümee

Nicht untersucht

Close

```
</einkapitel>  
</mehrerekapitel>
```

kapitel2.dtd:

```
<!-- kapitel2.dtd          -->  
<!-- passt zu kapitel2.xml -->  
<!ELEMENT mehrerekapitel (einkapitel)+ >  
<!ELEMENT einkapitel      (titel, rumpf) >  
<!ELEMENT titel           (#PCDATA)      >  
<!ELEMENT rumpf           (absatz)+       >  
<!ELEMENT absatz          (#PCDATA)      >
```

XML

PDF

Arbeitsabläufe

Resümee

Nicht ...

Close

adr6.xml:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!-- adr6.xml -->
<?xml-stylesheet href="adr2.xsl" type="text/xsl" ?>
<!DOCTYPE adressen SYSTEM "adressen.dtd" >
<adressen>

<adresse nr="1">
  <anrede>Frau</anrede>
  <titel>Dr.</titel>
  <name>
    <vorname>Anna</vorname>
    <nachname>Mayer</nachname>
  </name>
  <strasse>
    <sname>Blumenweg</sname>
    <hnr>2</hnr>
  </strasse>
  <ort>
    <plz>35392</plz>
    <ortsname>Gießen</ortsname>
```

XML

PDF

Arbeitsabläufe

Resümee

Nicht untersucht

Close

```
</ort>
<telefon>0641-123456</telefon>
<fax>0641-123459</fax>
<email>Anna.Mayer@t-online.de</email>
</adresse>
```

```
<adresse nr="2">
  <anrede>Herr</anrede>
  <name>
    <vorname>Emil</vorname>
    <nachname>Müller</nachname>
  </name>
  <strasse>
    <sname>Tulpenweg</sname>
    <hnr>22a</hnr>
  </strasse>
  <ort>
    <plz>35398</plz>
    <ortsname>Gießen</ortsname>
  </ort>
  <telefon>0641-234567</telefon>
```

XML

PDF

Arbeitsabläufe

Resümee

Nicht untersucht

Close


```
<email>Emil.Mueller@profanistik.uni-giessen.de</email>  
</adresse>
```

```
<adresse nr="3">  
  <anrede>Frau</anrede>  
  <titel>Prof. Dr.</titel>  
  <name>  
    <vorname>Erna</vorname>  
    <nachname>Schneider</nachname>  
  </name>  
  <strasse>  
    <sname>Bachstraße</sname>  
    <hnr>10</hnr>  
  </strasse>  
  <postfach>12345</postfach>  
  <ort>  
    <plz>12340</plz>  
    <ortsname>Nixdorf</ortsname>  
  </ort>  
  <telefon>09994-1234</telefon>  
  <fax>09994-12349</fax>  
  <email>Erna.Schneider@gmx.de</email>
```

[XML](#)[PDF](#)[Arbeitsabläufe](#)[Resümee](#)[Nicht untersucht](#)[Close](#)

</adresse>
</adressen>

XML

PDF

Arbeitsabläufe

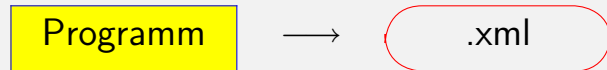
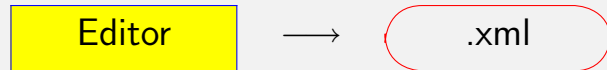
Resümee

Nicht ...

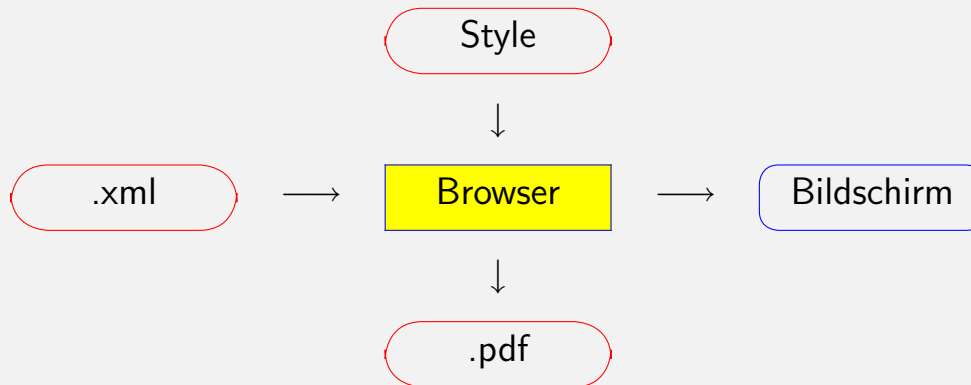
Close

3.1 XML

Erstellen von XML-Dateien:



Direkte Darstellung von XML-Dateien:



XML

PDF

Arbeitsabläufe

Resümee

Nicht untersucht

Close

Konvertieren von XML-Dateien:

Umsetzung mit Hilfe eigener, sehr spezieller Programme:



```
gawk -f zerteil.awk kapitel2.xml |  
gawk -f kap2t.awk > kapitel2.tex
```

⇒ **Zeigen!**

oder

```
gawk -f zerteil.awk kapitel2.xml |  
gawk -f kap2h.awk > kapitel2.html
```

⇒ **Zeigen!**

XML

PDF

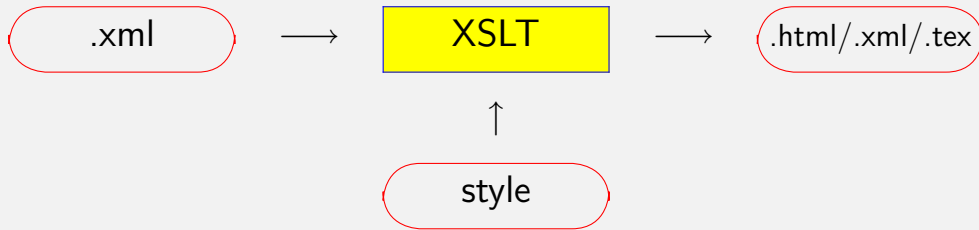
Arbeitsabläufe

Resümee

Nicht ...

Close

Umsetzung mit Hilfe von XSLT-Prozessoren:



```
saxon kapitel2.xml kapitel2-tex1.xsl > kapitel2a.tex
```

⇒ **Zeigen!**

oder

```
saxon kapitel2.xml kapitel2-html1.xsl > kapitel2a.html
```

⇒ **Zeigen!**

XML

PDF

Arbeitsabläufe

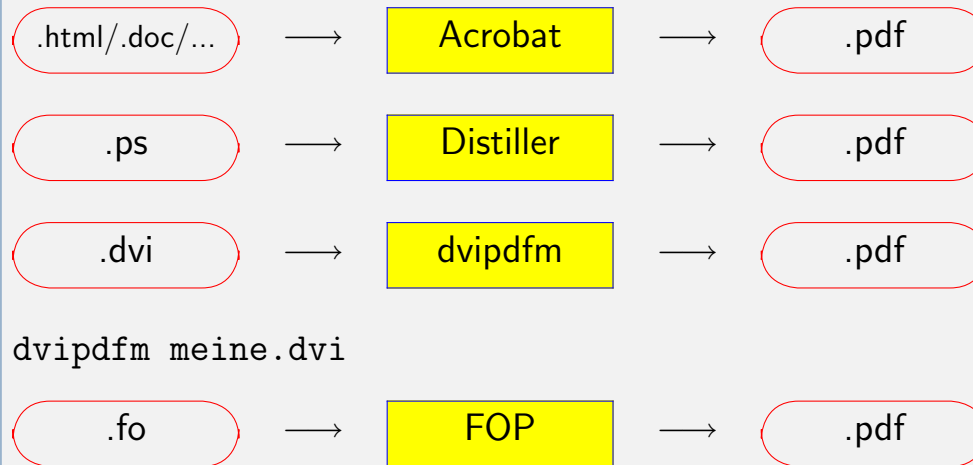
Resümee

Nicht ...

Close

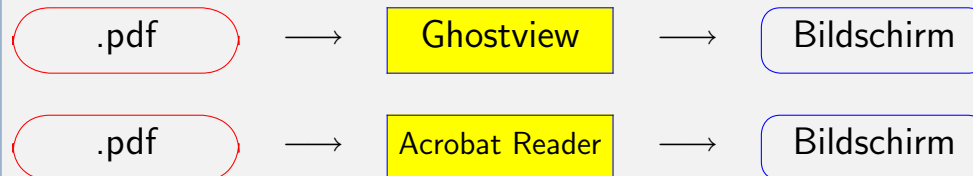
3.2 PDF

Erstellen von PDF-Dateien:



`dvipdfm meine.dvi`

Darstellen von PDF-Dateien:



XML

PDF

Arbeitsabläufe

Resümee

Nicht ...

Close

3.3 T_EX

Klassische Arbeitsabläufe bei T_EX:



```
pdflatex meine.tex  
latex meine.tex
```



```
dvips -o meine.ps meine.dvi
```



```
dvipdfm meine.dvi
```

⇒ **Zeigen!**

XML

PDF

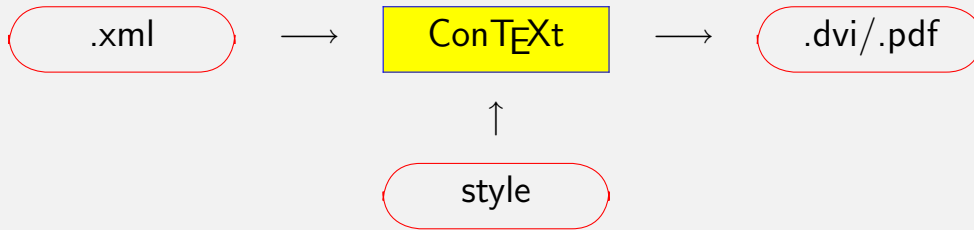
Arbeitsabläufe

Resümee

Nicht ...

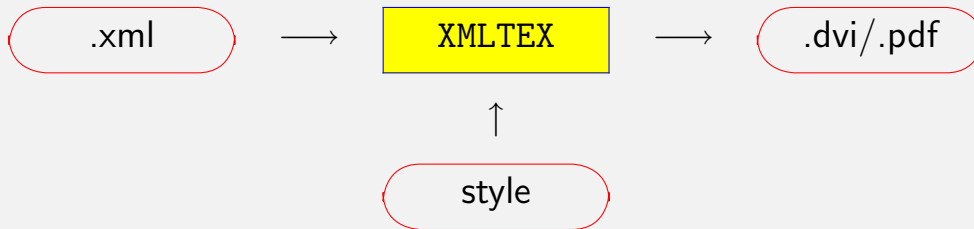
Close

Neuere Möglichkeiten:



```
texexec --pdf -env=tut7.dir adr6.xml
```

⇒ **Zeigen!**



```
pdfxmltex adr6.xml
```

⇒ **Zeigen!**

XML

PDF

Arbeitsabläufe

Resümee

Nicht ...

Close

.fo



PassiveT_EX



.dvi/.pdf

```
pdfxmltex meine.xml
```

XML

PDF

Arbeitsabläufe

Resümee

Nicht ...

Close

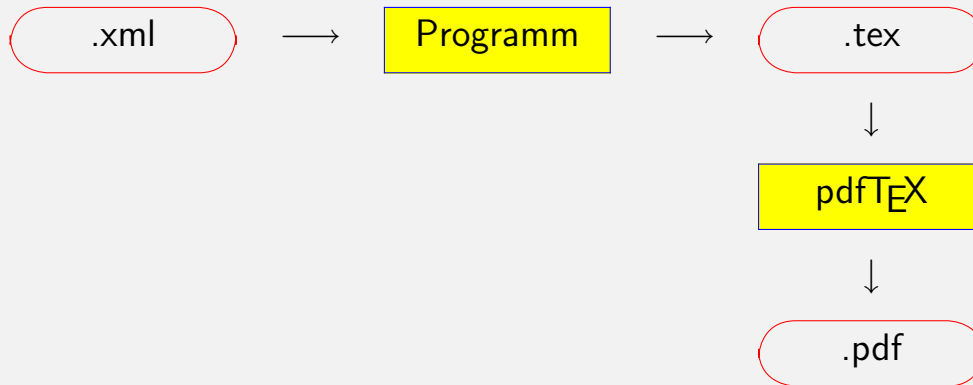
4 Resümee

Neben

1. ConT_EXt und
2. XML_TEX

ergeben sich damit die folgenden drei Arbeitsläufe von Interesse:

3. XML \Rightarrow Programm \Rightarrow pdfT_EX \Rightarrow PDF



```
gawk -f zerteil.awk kapitel2.xml |  
    gawk -f kap2t.awk > kapitel2.tex  
pdflatex kapitel2.tex
```

XML

PDF

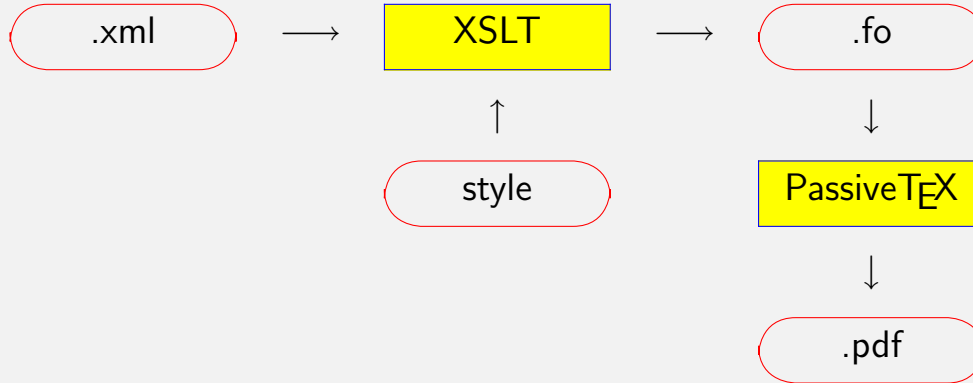
Arbeitsabläufe

Resümee

Nicht ...

Close

4. XML \Rightarrow XSLT \Rightarrow PassiveT_EX \Rightarrow PDF



```
saxon meine.xml meine.xsl > meine-fo.xml  
pdfxsltex meine-fo.xml
```

XML

PDF

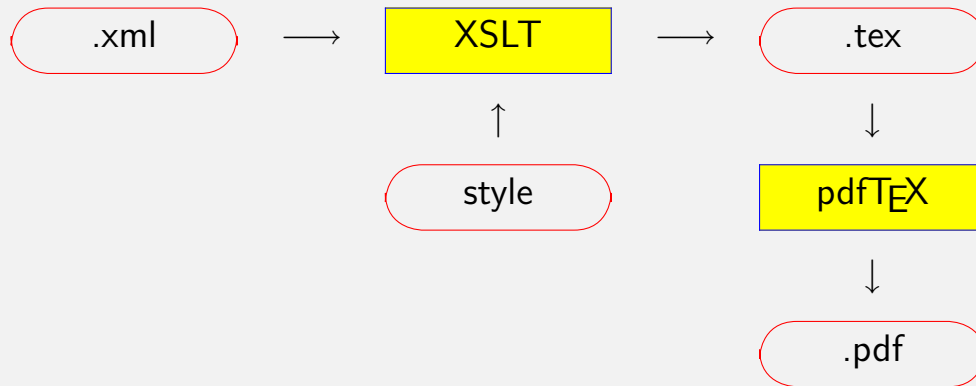
Arbeitsabläufe

Resümee

Nicht ...

Close

5. XML \Rightarrow XSLT \Rightarrow pdfT_EX \Rightarrow PDF



```
saxon kapitel2.xml kapitel2-tex1.xsl > kapitel2a.tex  
pdflatex kapitel2a.tex
```

XML

PDF

Arbeitsabläufe

Resümee

Nicht ...

Close

Kurz und bündig:

.xml	⇒	Programm	⇒	.tex	⇒	pdfT _E X	⇒	.pdf
.xml	⇒	XSLT	⇒	.tex	⇒	pdfT _E X	⇒	.pdf
.xml	⇒				⇒	XMLT _E X	⇒	.pdf
.xml	⇒	XSLT	⇒	.fo	⇒	PassiveT _E X	⇒	.pdf
.xml	⇒				⇒	ConT _E Xt	⇒	.pdf

XML

PDF

Arbeitsabläufe

Resümee

Nicht ...

Close

5 Nicht untersucht

JadeT_EX

- T_EX-Makro-Paket, um die Ausgabe von Jade/OpenJade (mit der Option `-t tex`) aufzubereiten
- Autor: Sebastian Rahtz
- Informationen: [JadeTeX](#)
- Download: [SourceForge: Project Info – jadetex](#)

XML

PDF

Arbeitsabläufe

Resümee

Nicht ...

Close

TEXML

- im Wesentlichen: Java-Programm `TEXMLatte.java`, das ein TeXML-Dokument nach $\text{T}_{\text{E}}\text{X}$ konvertiert, wobei TeXML eine spezielle XML-DTD ist
- Autor: alphaWorks (IBM)
- Informationen: [alphaWorks:TeXML](#)
- Download: [alphaWorks:TeXML:Download](#)
- TeXML wird offensichtlich nicht mehr weiter entwickelt.

XML

PDF

Arbeitsabläufe

Resümee

Nicht ...

Close

Von XML nach PDF über T_EX – ein detaillierter Blick auf zwei Verfahren

Günter Partosch, Oktober 2002

1 Übersicht

Aufbauend auf **Übersicht über die Verfahren** sollen jetzt zwei Verfahren näher untersucht werden:

- XML_{TEX} + Passive_{TEX}
- Con_{TEX}t

Übersicht

Close

Von XML nach PDF über T_EX – ein detaillierter Blick auf zwei Verfahren hier: XML und XMLT_EX/ PassiveT_EX

Günter Partosch, Oktober 2002

Übersicht:

- „XMLTEX ?“
- „Befehle“
- „Beispiele“
- „PassiveT_EX ?“
- „XSL Formatting Objects“

XMLTEX ?

Befehle

Beispiele

PassiveT_EX ?

XSL ...

Close

1 XMLTEX ?

Was ist XMLTEX?

XMLTEX ist ein T_EX/L^AT_EX-Makropaket zum Setzen von XML-Dateien (David Carlisle).

XMLTEX

- implementiert einen nicht-validierenden XML-Parser,
- beachtet die Konventionen (Recommendations) **Extensible Markup Language (XML) 1.0 (Second Edition)** und **Namespaces in XML**,
- verarbeitet Element-Inhalte, Element- und Attribut-Namen in UTF-8, ISO 8859-1, ISO 8859-2 oder in einem anderen brauchbaren 8-Bit-Code und
- setzt die analysierte XML-Datei mit Hilfe von L^AT_EX.

XMLTEX ?

Befehle

Beispiele

PassiveT_EX ?

XSL ...

Close

Voraussetzungen:

- eigentlich keine zusätzlichen, wenn die T_EX-Live7-Installation funktioniert

Installation:

- mit T_EX-Live7-CD-ROM: Wird automatisch erledigt.
- sonst: notwendige Dateien besorgen, entsprechende Formate generieren, File-Datenbank aktualisieren

Konfiguration:

Das meiste geschieht in der Konfigurationsdatei `xmltex.cfg`, beispielsweise:

```
\SYSTEM{http://www.oucs.ox.ac.uk/dtds/tei-oucs.dtd}{tei.xmt}  
\NAMESPACE{http://www.w3.org/1999/XSL/Format}{fotex.xmt}  
\NAMESPACE{http://www.w3.org/1998/Math/MathML}{mathml2.xmt}  
\NAMESPACE{http://www.dcarlisle.demon.co.uk/sec}{sec.xmt}  
\NAME{langtest}{langtest.xmt}
```

XMLT_EX ?

Befehle

Beispiele

PassiveT_EX ?

XSL ...

Close

```
\NAME{TEI.2}{tei.xmt}
```

```
\NAME{html}{html.xmt}
```

oder

```
\UnicodeCharacter{160}{\nobreakspace}
```

```
\UnicodeCharacter{161}{\textexclamdown }
```

```
\UnicodeCharacter{162}{\ifmmode\mbox{\textcent}\else\textcent\fi}
```

```
\UnicodeCharacter{163}{\ifmmode \pounds \else \textsterling \fi}
```

XMLTEX ?

Befehle

Beispiele

PassiveT_EX ?

XSL ...

Close

Regeln:

- Die Konfigurationsdatei `xmltex.cfg` wird bei jedem Aufruf von XML-TEX ausgewertet.
- Zusätzlich kann die Datei `datei.cfg` bereit gestellt werden (wird beim Verarbeiten der Datei `datei.xml` beachtet).
- Die Dateien der Art `*.xmt` beschreiben, wie die XML-Elemente zu interpretieren und zu setzen sind.

Arbeitsweise:

sehr verkürzt – und hoffentlich nicht allzu (!) falsch:

- Bei einem Aufruf von XMLTEX für die Datei `datei.xml` wird die Konfigurationsdatei `xmltex.cfg` gelesen.
- Falls vorhanden, wird auch die Datei `datei.cfg` geladen.
- In beiden Fällen speichert XMLTEX die in den Anweisungen `\NAMESPACE`, `\PUBLIC`, `\SYSTEM`, `\NAME` und `\XMLNS` vereinbarten Begriffe.

XMLTEX ?

Befehle

Beispiele

PassiveTEX ?

XSL ...

Close

- Beim Parsen analysiert XMLTEX die XML-Datei, welche der obigen Begriffe zutreffen.
- XMLTEX lädt die betreffenden XMLTEX-Pakete.
- XMLTEX setzt die XML-Datei.

Aufruf (wenn XMLTEX-Formate generiert wurden)

`xmltex xml-datei`

oder

`pdfxmltex xml-datei`

Informationen:

- **xmltex: A non validating (and not 100% conforming) namespace aware XML parser implemented in TeX**
(<ftp://ftp.dante.de/tex-archive/macros/xmltex/base/manual.html>)
- David Carlisle: *XMLTEX: A non validating (and not 100% conforming) namespace aware XML parser implemented in T_EX*; TUGBoat, 3/2000, pp. 193–199

XMLTEX ?

Befehle

Beispiele

PassiveT_EX ?

XSL ...

Close

2 Befehle

- „Befehle in Konfigurationsdateien (Auswahl)“
- „Wichtige Befehle in XML_{TE}X-Paketen“

XML_{TE}X ?

Befehle

Beispiele

Passive_{TE}X ?

XSL ...

Close

2.1 Befehle in Konfigurationsdateien (Auswahl)

- `\NAMESPACE{URI}{xmt-file}`
- `\PUBLIC{FPI}{file}`
- `\SYSTEM{URI}{file}`
- `\NAME{element-name}{xmt-file}`
- `\XMLNS{element-name}{URI}`
lädt die L^AT_EX-Kommandos aus der Datei *URI*, wenn *element-name* in der XML-Datei erkannt wird??
- `\xmltraceonly`
brauchbar bei der Fehlersuche: XMLTEX stoppt die Aufbereitung des Dokuments.
- `\xmltraceoff`
In der Voreinstellung protokolliert XMLTEX jeden Beginn und Schluss eines Elements. Durch `\xmltraceoff` kann dieses Verhalten abgestellt werden.

XMLTEX ?

Befehle

Beispiele

PassiveT_EX ?

XSL ...

Close

- `\inputonce{xmt-file}`
veranlasst, dass das XML-Paket *xmt-file* immer geladen wird
- `\UnicodeCharacter{hex-or-dec}{tex-code}`
assoziiert das Unicode-Zeichen *hex-or-dec* mit dem T_EX-Code *tex-code*
- `\ActivateASCII{hex-or-dec}`
macht das Zeichen *hex-or-dec* aktiv

XMLTEX ?

Befehle

Beispiele

PassiveT_EX ?

XSL ...

Close

2.2 Wichtige Befehle in XMLTEX-Paketen

- `\FileEncoding{encoding}`
legt lokal das Encoding fest; Voreinstellung ist UTF-8
- `\DeclareNamespace{prefix}{URI}`
assoziiert lokal den Präfix *prefix* mit *URI*
- `\XMLelement{element-name}{attribute-spec}{begin-code}{end-code}`

analog zur \LaTeX -Anweisung `\newenvironment`:

- legt in *begin-code* bzw. *end-code* fest, welche \LaTeX -Anweisungen am Anfang bzw. Ende des Elements *element-name* ausgeführt werden sollen
- vereinbart mit Hilfe von `\XMLattribute` in *attribute-spec* Attribute und ihre Voreinstellungen
- `\XMLattribute{attribute-qname}{command-name}{default}`
kann nur innerhalb von `\XMLelement` benutzt werden:

XMLTEX ?

Befehle

Beispiele

Passive \TeX ?

XSL ...

Close

- erstes Argument: Attribut *attribute-qname* wird spezifiziert
- zweites Argument: vereinbart die L^AT_EX-Anweisung *command-name*; sie enthält den Wert des Attributs
- drittes Argument: Voreinstellung des Attributs

- `\XMLelement{element-qname}{attribute-spec}{\xmlgrab}{end-code}`

Spezialfall: In *end-code* kann auf den Inhalt des Elements *element-qname* über #1 zugegriffen werden.

- `\XMLentity{name}{code}`

analog zu der Deklaration `<!ENTITY...>`: Entity *name* wird mit dem L^AT_EX-Code *code* assoziiert.

- `\XMLname{name}{command-name}`

vereinbart das L^AT_EX-Kommando *command-name*: enthält den normalisierten XML-Namen *name*; kann z.B. in Vergleichsanweisungen der Art

`\ifx\xml@parent` verwendet werden

- `\XMLstring{command-name}<>XML Data</>`

speichert den XML-Code *XML Data* im L^AT_EX-Kommando *command-name*

XMLTEX ?

Befehle

Beispiele

PassiveT_EX ?

XSL ...

Close

Ausführliche Beschreibungen aller derartiger Befehle findet man in **xmltex**:
A non validating (and not 100% conforming) namespace aware
XML parser implemented in TeX (Handbuch).

XMLTEX ?

Befehle

Beispiele

PassiveT_EX ?

XSL ...

Close

3 Beispiele

Beispiel 1:

XML-Datei adr6.xml:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!-- adr6.xml -->
<?xml-stylesheet href="adr2.xsl" type="text/xsl" ?>
<!DOCTYPE adressen SYSTEM "adressen.dtd" >
<adressen>

<adresse nr="1">
  <anrede>Frau</anrede>
  <titel>Dr.</titel>
  <name>
    <vorname>Anna</vorname>
    <nachname>Mayer</nachname>
  </name>
  <strasse>
```

XMLTEX ?

Befehle

Beispiele

PassiveT_EX ?

XSL Formatting Objects

Close

```
<sname>Blumenweg</sname>
<hnr>2</hnr>
</strasse>
<ort>
  <plz>35392</plz>
  <ortsname>Gießen</ortsname>
</ort>
<telefon>0641-123456</telefon>
<fax>0641-123459</fax>
<email>Anna.Mayer@t-online.de</email>
</adresse>
```

```
<adresse nr="2">
  <anrede>Herr</anrede>
  <name>
    <vorname>Emil</vorname>
    <nachname>Müller</nachname>
  </name>
  <strasse>
    <sname>Tulpenweg</sname>
    <hnr>22a</hnr>
  </strasse>
```

XMLTEX ?

Befehle

Beispiele

PassiveT_EX ?

XSL Formatting Objects

Close


```
<ort>
  <plz>35398</plz>
  <ortsname>Gießen</ortsname>
</ort>
<telefon>0641-234567</telefon>
<email>Emil.Mueller@profanistik.uni-giessen.de</email>
</adresse>
```

```
<adresse nr="3">
  <anrede>Frau</anrede>
  <titel>Prof. Dr.</titel>
  <name>
    <vorname>Erna</vorname>
    <nachname>Schneider</nachname>
  </name>
  <strasse>
    <sname>Bachstraße</sname>
    <hnr>10</hnr>
  </strasse>
  <postfach>12345</postfach>
  <ort>
    <plz>12340</plz>
```

XMLTEX ?

Befehle

Beispiele

PassiveT_EX ?

XSL Formatting Objects

Close

```
<ortsname>Nixdorf</ortsname>
</ort>
<telefon>09994-1234</telefon>
<fax>09994-12349</fax>
<email>Erna.Schneider@gmx.de</email>
</adresse>
</adressen>
```

Beim Verarbeiten werden der Reihe nach die Dateien `xmltex.cfg`, `adr6.cfg`, `iso-8859-1.xmt`, `adressen.xmt` geladen.

Konfigurationsdatei `adr6.cfg`:

```
\NAME{adressen}{adressen.xmt}
```

XMLTEX ?

Befehle

Beispiele

PassiveTEX ?

XSL ...

Close

XMLTEX-Paket adressen.xmt:

```
\XMLElement{adressen}{}
  {\documentclass[10pt,a4paper]{article}
  \usepackage{adr-tex}
  \title{Adressen}
  \author{Emil Mayer}
  \date{\today}
  \begin{document}
  \maketitle
  \section*{Adressen}
  }
  {\end{document}}

\xMLElement{adresse}{\XMLAttribute{nr}{\nummer}{--}}
  {\begin{adresse}{\nummer}}
  {\end{adresse}}

\xMLElement{anrede}    {} {\xmlgrab} {\anrede{#1}}
\xMLElement{titel}     {} {\xmlgrab} {\titel{#1}}
\xMLElement{vorname}   {} {\xmlgrab} {\vorname{#1}}
\xMLElement{nachname}  {} {\xmlgrab} {\nachname{#1}}
\xMLElement{sname}     {} {\xmlgrab} {\sname{#1}}
```

XMLTEX ?

Befehle

Beispiele

PassiveT_EX ?

XSL Formatting Objects

Close

```
\XMLElement{hnr}      {} {\xmlgrab} {\hnr{#1}}  
\XMLElement{postfach} {} {\xmlgrab} {\postfach{#1}}  
\XMLElement{plz}       {} {\xmlgrab} {\plz{#1}}  
\XMLElement{ortsname}  {} {\xmlgrab} {\ortsname{#1}}  
\XMLElement{telefon}   {} {\xmlgrab} {\telefon{#1}}  
\XMLElement{fax}        {} {\xmlgrab} {\fax{#1}}  
\XMLElement{email}     {} {\xmlgrab} {\email{#1}}
```

⇒ Ergebnis

XMLTEX ?

Befehle

Beispiele

PassiveT_EX ?

XSL ...

Close

Beispiel 2:

Datei manual.xml aus der T_EX-Live7-Distribution
(TeXLive\TeXMF\DOC\XMLTEX\BASE):

Erste Zeilen der Datei:

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

<!DOCTYPE TEI.2 SYSTEM
    "http://www.oucs.ox.ac.uk/dtds/tei-oucs.dtd" [
<!NOTATION URL SYSTEM "" >
<!ENTITY lpp1 SYSTEM
    "http://www.latex-project.org/lpp1.txt" NDATA URL>
<!ENTITY ldots "&#x2026;">
<!ENTITY TeX "TeX">
<!ENTITY LaTeX "LaTeX">
]>

<TEI.2>
  <teiHeader>
    <fileDesc>
      <titleStmt>
```

XMLTEX ?

Befehle

Beispiele

PassiveT_EX ?

XSL ...

Close

```
<title>xmlltex</title>
</titleStmt>
<publicationStmt>
  <availability>
    <p>
```

This file is distributed under the LaTeX Project Public License (LPPL) as found at `<xptr doc="lppl"/>`.

Either version 1.0, or at your option, any later version.

```
</p>      </availability>
</publicationStmt>
```

...

Beim Verarbeiten der Datei werden der Reihe nach xmlltex.cfg und tei.xmt aufgerufen.

XMLTEX ?

Befehle

Beispiele

PassiveTEX ?

XSL ...

Close

Ausschnitte aus der Datei tei.xmt:

...

```
\XMLElement{TEI.2}{}  
  { \documentclass{article}  
    \usepackage{ifthen,url}  
    \begin{document}  }  
  {\end{document}}
```

```
\XMLElement{p}{}  
  {\par}  
  {}
```

...

```
\XMLElement{div}{  
  \XMLAttribute{id}{\idval}{\@nil}}  
  {\advance\SCOUNT\@ne}  
  {}
```

...

```
\XMLElement{emph}{}  
  {\itshape}  
  {}
```

...

XMLTEX ?

Befehle

Beispiele

PassiveT_EX ?

XSL ...

Close

⇒ Ergebnis

XMLTEX ?

Befehle

Beispiele

PassiveT_EX ?

XSL ...

Close

4 PassiveT_EX ?

Einige Anmerkungen zu PassiveT_EX:

- T_EX-Makro-Library
- Autor: Sebastian Rahtz
- PassiveT_EX realisiert die Standards für **Extensible Stylesheet Language (XSL) Version 1.0** und **Mathematical Markup Language (MathML) 1.01 Specification**,
- liest XML-Dokumente, die XSL-FOs (Formatting Objects) und -Elemente enthalten und benutzt L^AT_EX zum Setzen des Dokuments,
- basiert auf XMLT_EX und
- wurde aus JadeT_EX entwickelt.

XMLT_EX ?

Befehle

Beispiele

PassiveT_EX ?

XSL ...

Close

- Informationen:
 - **Text Encoding Initiative – PassiveTeX**
 - Sebastian Rahtz (DANTE'2001):
Future of TeX in a XML-dominated world
 - Michel Goossens, Sebastian Rahtz: *PassiveTeX: from XML to PDF*; TUGBoat, 3/2000, pp. 222–234
- Download: **PassiveTeX-Download**
- Beispiele für FO-erzeugende XSL-Style-Dateien:
Text Encoding Initiative: XSL stylesheets for TEI XML

Voraussetzungen:

- eigentlich keine zusätzlichen, wenn die TeX-Live7-Installation funktioniert

XMLTEX ?

Befehle

Beispiele

PassiveTeX ?

XSL ...

Close

Installation:

- mit T_EX-Live7-CD-ROM: Wird automatisch erledigt.
- sonst: notwendige Dateien besorgen, auspacken, File-Datenbank aktualisieren; setzt funktionierende XML_TEX-Installation voraus

XML_TEX ?

Befehle

Beispiele

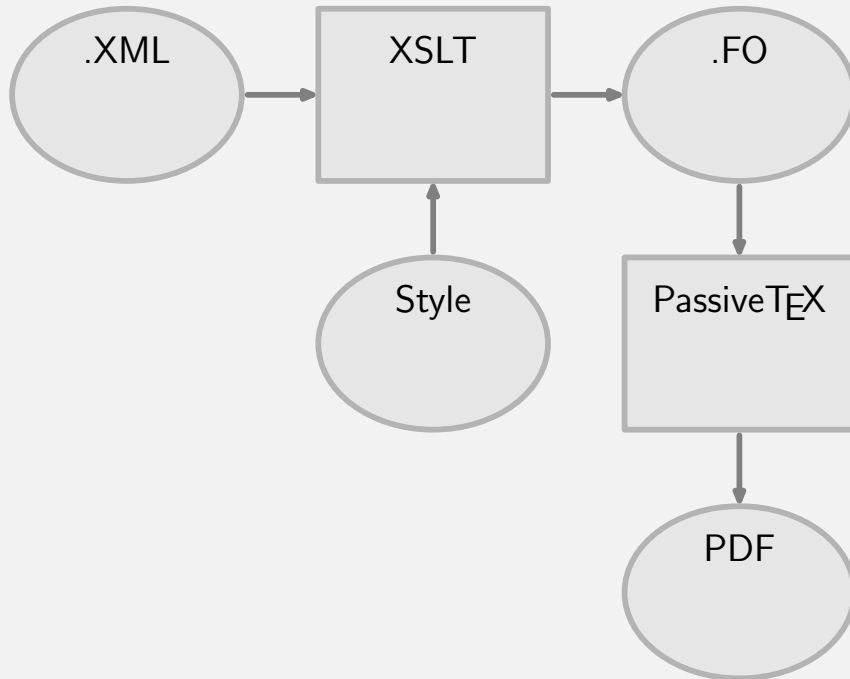
PassiveT_EX ?

XSL ...

Close

Arbeitsablauf:

```
saxon teiu5.xml tei.xsl > teiu5-fo.xml  
pdfxsltex teiu5-fo.xml
```



XMLTEX ?

Befehle

Beispiele

PassiveTeX ?

XSL ...

Close

5 XSL Formatting Objects

- Mit Hilfe von XSL Formatting Objects wird relativ detailliert Layout und Formatierung eines zu setzenden Dokuments festgelegt.
- Dateien mit Formatting Objects sind XML-Dateien. Sie werden normalerweise durch XSLT-Prozessoren erzeugt.
- FO-Prozessoren lesen diese Dateien und setzen das Dokument.
- Beispiele für FO-Prozessoren:
 - **FOP** aus dem *Apache XML Project*
 - PassiveT_EX

XMLT_EX ?

Befehle

Beispiele

PassiveT_EX ?

XSL ...

Close

Beispiel 1 (Äußeres Gerüst):

Das folgende XSLT-Fragment könnte Formatting Objects erzeugen, die die globalen Formatierungseigenschaften eines Dokuments festlegen:

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/XSL/Transform/1.0"
                xmlns:fo="http://www.w3.org/XSL/Format/1.0"
                default-space="">
  <xsl:variable name="PageMarginTop">75pt</xsl:variable>
  <xsl:variable name="PageMarginBottom">125pt</xsl:variable>
  <xsl:variable name="PageMarginLeft">80pt</xsl:variable>
  <xsl:variable name="PageMarginRight">150pt</xsl:variable>
  <xsl:variable name="BodySize">12pt</xsl:variable>

  <xsl:template match='/'>
    <fo:root xmlns:fo="http://www.w3.org/XSL/Format/1.0">
      <fo:layout-master-set>
        <fo:simple-page-master
          page-master-name="allpages"
          margin-top="{ $PageMarginTop }"
          margin-bottom="{ $PageMarginBottom }"
          margin-left="{ $PageMarginLeft }"
          margin-right="{ $PageMarginRight }"
          body-size="{ $BodySize }">
```

XMLTEX ?

Befehle

Beispiele

PassiveTEX ?

XSL ...

Close

```
margin-right="{ $PageMarginRight }">
  <fo:region-body margin-bottom="100pt"/>
  <fo:region-after extent="25pt"/>
</fo:simple-page-master>
</fo:layout-master-set>
<fo:page-sequence>
  <fo:sequence-specification>
    <fo:sequence-specifier-repeating
      page-master-first="allpages"
      page-master-repeating="allpages"/>
  </fo:sequence-specification>
  <fo:flow font-family="serif">
    <xsl:apply-templates/>
  </fo:flow>
</fo:page-sequence>
</fo:root>
</xsl:template>

...
</xsl:stylesheet>
```

XMLTEX ?

Befehle

Beispiele

PassiveTEX ?

XSL ...

Close

Beispiel 2 (Aufzählung):

XML-Fragment aus einem XML-Dokument:

```
<ol>
<li>Punkt 1</li>
<li>Punkt 2</li>
</ol>
```

Die zugehörigen Formatting Objects könnten (verkürzt) durch das folgende XSLT-Fragment generiert werden:

```
<xsl:template match="ol">
  <fo:list-block space-before.optimum="4pt">
    <xsl:apply-templates />
  </fo:list-block>
</xsl:template>
```

und

XMLTEX ?

Befehle

Beispiele

PassiveT_EX ?

XSL ...

Close


```
<xsl:template match="li">
  <fo:list-item>
    <fo:list-item-label>...</fo:list-item-label>
    <fo:list-item-body>
      <fo:block font-weight="normal">
        <xsl:apply-templates />
      </fo:block>
    </fo:list-item-body>
  </fo:list-item>
</xsl:template>
```

XMLTEX ?

Befehle

Beispiele

PassiveT_EX ?

XSL ...

Zurück zu

Detaillierter Blick auf zwei Verfahren
Übersicht über die Verfahren

Close

Von XML nach PDF über T_EX – ein detaillierter Blick auf zwei Verfahren hier: XML und ConT_EXt

Günter Partosch, Oktober 2002

Übersicht:

- „ConT_EXt ?“
- „texexec.pl ?“
- „Wege nach XML“
- „Abbildungen“
- „Beispiele“
- „Schon bekannt“

ConT_EXt ?

texexec.pl ?

Wege nach ...

Abbildungen

Beispiele

Schon bekannt

Close

1 ConT_EXt ?

Was ist ConT_EXt?

ein umfangreiches T_EX-Makropaket; entwickelt 1997–2002 bei Pragma ADE

(Hans Hagen, Ton Otten):

- ConT_EXt umfasst zusätzlich: Perl- und Perl/Tk-Programme; META-POST-Makros
- ConT_EXt passt zu den gängigen T_EX-Distributionen.
- Es ist hervorragend geeignet zum Setzen komplexer Dokumente mit hohen typographischen Anforderungen.
- ConT_EXt unterstützt das Erstellen hochwertiger – auch interaktiver – PDF-Dokumente.
- Layout und Aufbau der Ausgabe sind verhältnismäßig einfach durch zentrale Parameter steuerbar.
- Farbe und Grafik sind integriert.
- XML-Code kann interpretiert und dargestellt werden.

ConT_EXt ?

texexec.pl ?

Wege nach ...

Abbildungen

Beispiele

Schon bekannt

Close

Einstiegsinformationen:

- **Read Me First**
- **ConT_EXt an excursion**

Voraussetzungen:

- neuer T_EX-Prozessor mit PDF-Unterstützung (pdfT_EX; besser noch pdf- ϵ T_EX)
- METAPOST
- Perl

Installation:

- mit T_EXLive7-CD-ROM: Wird automatisch erledigt.
- sonst:
 - neueste **offizielle ConT_EXt-Distribution**
(<http://www.pragma-ade.nl/context/cont-tfm.zip>) besorgen
 - entpacken

ConT_EXt ?

texexec.pl ?

Wege nach ...

Abbildungen

Beispiele

Schon bekannt

Close

- konfigurieren
- neues Format erzeugen
- Informationen zur Installation:
 - **how to install ConT_EXt**
 - **T_EXEXEC explained**
 - **T_EXUTIL explained**

Weitere Informationen:

- Einstiegspunkt für alle weiteren Informationen bezüglich ConT_EXt:
PRAGMA Advanced Document Engineering
(<http://www.pragma-ade.nl/>)
- spezieller Einstiegspunkt für weitere Informationen bezüglich ConT_EXt & XML:
The ConT_EXt XML Page (<http://www.pragma-ade.nl/xml.htm>);
darunter:
 - **XML in ConT_EXt**
 - **CHEMML**
 - **FIGURES**
 - **MATHML**
 - **PHYSML**
 - **STEPS**

ConT_EXt ?

texexec.pl ?

Wege nach ...

Abbildungen

Beispiele

Schon bekannt

Close

Hilfreich ist auch

- die **Diskussionsliste** `ntg-context@ntg.nl` .

ConT_EXt ?

`texexec.pl` ?

Wege nach ...

Abbildungen

Beispiele

Schon bekannt

Close

2 texexec.pl ?

texexec.pl ist ein unentbehrliches Werkzeug beim Arbeiten mit ConT_EXt & XML:

- Aufgabe: texexec.pl
 - analysiert Log-Dateien und sorgt automatisch für die richtige Anzahl von Durchläufen,
 - ermöglicht den zusätzlichen Einsatz von Modulen, Direktivendateien oder XML-Filter,
 - arrangiert die Ausgabe neu oder bereitet sie nach,
 - generiert auf Anfrage ConT_EXt-Formate,
 - ...
- Voraussetzungen: neue ConT_EXt-Installation mit Perl
- Installation: automatisch mit T_EXLive7-CD-ROM; ggf. noch Anpassungen für Perl und in der texexec-Konfigurationsdatei texexec.ini

ConT_EXt ?

texexec.pl ?

Wege nach ...

Abbildungen

Beispiele

Schon bekannt

Close

- Informationen: [T_EXEXEC explained](#); außerdem in [how to install ConT_EXt](#)

- Aufruf und Parameter:

textexec optionen datei

Einige wichtige Optionen (aus der Ausgabe bei `--help`):

<code>--usemodule</code>	load some modules first
<code>--xmlfilter</code>	apply XML filter
<code>--environment</code>	load some environments first
<code>--listing</code>	produce a verbatim listing
<code>--make</code>	build format files
<code>--mode</code>	running mode
<code>--output</code>	specials to use
<code>--pages</code>	pages to output
<code>--paper</code>	paper input and output format
<code>--path</code>	document source path
<code>--pdfarrange</code>	arrange pdf pages
<code>--pdfcombine</code>	combine pages to one page
<code>--pdfcopy</code>	scale pages down/up
<code>--pdfselect</code>	select pdf pages

ConT_EXt ?

texexec.pl ?

Wege nach XML

Abbildungen

Beispiele

Schon bekannt

Close

`--pdf` produce PDF directly using pdf(e)tex
`--suffix` resulting file suffix
`--runs` maximum number of TeX runs
`--verbose` shows some additional info

- Aufruf-Beispiele:

- `texexec datei1.tex`
- `texexec --pdf datei2.tex`
- `texexec --pdf --paper=a4a3 datei3.tex`
- `texexec --pdf --pages=even datei4.tex`
- `texexec --pdf --env=mein.dir datei5.xml`
- `texexec --pdf --xmlfilter=mml,mmp,mmc datei6.xml`
- `texexec --pdf --usemodule=mathml,steps datei7.xml`

ConT_EXt ?

texexec.pl ?

Wege nach ...

Abbildungen

Beispiele

Schon bekannt

Close

3 Wege nach XML

ConT_EXt verarbeitet

- XML-Code, der in ein Dokument eingebettet ist bzw.
- vollständige XML-Dateien.

Folgende Methoden stehen u.a. zur Verfügung:

- „**Pretty Printing**: `definetyping`, `setuptyping`“
- „`enableXML` und **Processing Instruction (1)**“
- „`enableXML` und **Processing Instruction (2)**“
- „`startbuffer ... stopbuffer`, `typebuffer`, ...“
- „`startXMLdata ... stopXMLdata`“
- „`startXMLcode ... stopXMLcode`, `getXMLcode`“
- „`processXMLfilegrouped`“
- „`texexec`“

ConT_EXt ?

`texexec.pl` ?

Wege nach ...

Abbildungen

Beispiele

Schon bekannt

Close

3.1 Pretty Printing: definetyping, setuptyping

```
\definetyping[meinXML][mode=XML, ...]  
\setuptyping[meinXML][ ... ]  
...  
\startmeinXML  
...XMLCode  
\stopmeinXML
```

ConT_EXt ?

texexec.pl ?

Wege nach ...

Abbildungen

Beispiele

Schon bekannt

Close

3.2 enableXML und Processing Instruction (1)

```
\defineXMLgrouped [name] {\kommando}  
...  
\enableXML  
  
...<name>elementinhalt</name> ...  
...  
<?context-command \disableXML ?>
```

ConT_EXt ?

texexec.pl ?

Wege nach ...

Abbildungen

Beispiele

Schon bekannt

Close

piel:

```
% tutc1.tex
\mainlanguage[de]
\language[de]

% (3)
% \defineXMLgrouped [betont] {\it }

\starttext
normaler Text -- {\bf fetter Text} -- normaler Text --
normaler Text -- normaler Text -- normaler Text --
normaler Text -- normaler Text -- normaler Text --
normaler Text -- normaler Text --

normaler Text -- mit k"urzerem
% (1)
%\enableXML
<betont>XML-Einsprengsel</betont>
%<!-- (2) -->
%<!--<?context-command \disableXML ?> -->
-- normaler Text --
```

ConT_EXt ?

texexec.pl ?

Wege nach XML

Abbildungen

Beispiele

Schon bekannt

Close

```
normaler Text -- {\bf fatter Text} -- normaler Text --  
normaler Text -- normaler Text -- normaler Text --  
normaler Text -- normaler Text --
```

```
normaler Text -- normaler Text -- normaler Text --  
normaler Text -- normaler Text -- normaler Text --  
normaler Text -- normaler Text -- normaler Text --  
normaler Text -- normaler Text --
```

```
\stoptext
```

Nach den sukzessiven Änderungen bei (1), (2) und (3) ergibt sich als endgültiges **Ergebnis**.

ConT_EXt ?

texexec.pl ?

Wege nach ...

Abbildungen

Beispiele

Schon bekannt

Close

3.3 enableXML und Processing Instruction (2)

```
\defineXMLgrouped [name] {\kommando}  
...  
\starttext  
\enableXML  
  
...  
...<name>elementinhalt</name> ...  
...  
<?context-command \stoptext ?>
```

ConT_EXt ?

texexec.pl ?

Wege nach ...

Abbildungen

Beispiele

Schon bekannt

Close

piel:

```
% tutc2.tex
\mainlanguage[de]
\language[de]

\defineXMLgrouped [betont] {\it }

\starttext
\enableXML
<betont>XML-Einsprengsel</betont>
<?context-command \stoptext ?>
-- normaler Text --
normaler Text -- {\bf fetter Text} -- normaler Text --
normaler Text -- normaler Text -- normaler Text --
normaler Text -- normaler Text --
```

⇒ Ergebnis

ConT_EXt ?

texexec.pl ?

Wege nach ...

Abbildungen

Beispiele

Schon bekannt

Close

3.4 startbuffer ... stopbuffer, typebuffer, processXMLbuffer

```
...  
\startbuffer  
...XMLCode  
\stopbuffer  
...  
\typebuffer  
...  
\processXMLbuffer  
...
```

ConT_EXt ?

texexec.pl ?

Wege nach ...

Abbildungen

Beispiele

Schon bekannt

Close

Kleines Beispiel (MathML, useXMLfilter, startbuffer, processbuffer):

```
%ut3.tex
\useXMLfilter[mml,mmp,mmc]
\starttext
\startbuffer
<math>
  <apply><eq />
    <apply><power /> <ci>a</ci> <ci>2</ci> </apply>
    <apply><plus />
      <apply><power /> <ci>b</ci> <ci>2</ci> </apply>
      <apply><power /> <ci>c</ci> <ci>2</ci> </apply>
    </apply>
  </apply>
</math>
\stopbuffer

\typebuffer
\processXMLbuffer
\stoptext
```

⇒ Ergebnis

ConT_EXt ?

texexec.pl ?

Wege nach ...

Abbildungen

Beispiele

Schon bekannt

Close

3.5 startXMLdata ... stopXMLdata

```
...  
\startXMLdata  
...XMLCode  
\stopXMLdata  
...
```

ConT_EXt ?

texexec.pl ?

Wege nach ...


Abbildungen

Beispiele

Schon bekannt

Close

Beispiel (MathML, useXMLfilter, startXMLdata):

```
%ut4.tex
\useXMLfilter[mml,mmp,mmc]
\starttext
\startXMLdata
<math>
  <apply><sin />
    <apply><plus /><ci>a</ci> <ci>b</ci></apply>
  </apply>
</math>
\stopXMLdata

\stoptext
```

⇒ Ergebnis

ConT_EXt ?

texexec.pl ?

Wege nach ...

Abbildungen

Beispiele

Schon bekannt

Close

3.6 startXMLcode ... stopXMLcode, getXMLcode

```
...  
\startXMLcode[name]  
...XMLCode  
\stopXMLcode  
...  
\getXMLcode[name]  
...
```

ConT_EXt ?

texexec.pl ?

Wege nach ...

Abbildungen

Beispiele

Schon bekannt

Close

Beispiel (MathML, useXMLfilter, startXMLcode):

```
%≡ut5.tex
\useXMLfilter[mml,mmp,mmc]
\starttext
\startXMLcode[hugo]
<math>
  <mrow>
    <mrow>
      <msup><mi>x</mi><mn>2</mn></msup>
      <mo>+</mo>
      <mrow>
        <mn>4</mn>
        <mo>&InvisibleTimes;</mo>
        <mi>x</mi>
      </mrow>
      <mo>+</mo><mn>4</mn>
    </mrow>
    <mo>=</mo><mn>0</mn>
  </mrow>
</math>
\stopXMLcode
```

ConT_EXt ?

texexec.pl ?

Wege nach XML

Abbildungen

Beispiele

Schon bekannt

Close

```
\startXMLcode[erna]
<math>
  <apply><plus /> <ci>a</ci> <ci>b</ci> <ci>c</ci> </apply>
</math>
\stopXMLcode
```

```
\getXMLcode[hugo]
\getXMLcode[erna]
```

```
\stoptext
```

⇒ Ergebnis

ConT_EXt ?

texexec.pl ?

Wege nach ...

Abbildungen

Beispiele

Schon bekannt

Close

3.7 processXMLfilegrouped

```
...  
...abbildungsdirektiven  
...  
\starttext  
\processXMLfilegrouped{xmldatei}  
\stoptext
```

Außerdem noch

```
\processXMLfile{xmldatei}
```

ConT_EXt ?

texexec.pl ?

Wege nach ...

Abbildungen

Beispiele

Schon bekannt

Close

piel:

```
% tutc6.tex
\language [de]

\defineXMLenvironment [einkapitel] {\starttext} {\stoptext}
\defineXMLenvironment [absatz]     {}           {\par}
\defineXMLargument    [titel]      {\chapter}

\starttext
\processXMLfilegrouped{kapitel1.xml}
\stoptext
```

⇒ Ergebnis

ConT_EXt ?

texexec.pl ?

Wege nach ...

Abbildungen

Beispiele

Schon bekannt

Close

3.8 texexec

Eleganteste Möglichkeit, XML-Dateien mit ConT_EXt zu verarbeiten / darzustellen:

Aufruf mit Direktivendatei (environment file):

```
texexec --pdf --env=direktivendatei xmldatei
```

Aufruf mit XML-Filter:

```
texexec --pdf --xmlfilter=xml-filter xmldatei
```

Aufruf-Beispiele:

- `texexec --pdf anfang.xml`
- `texexec --pdf --env=regeln weiter.xml`
- `texexec --pdf --xmlfilter=mml,mmp,mmc mathe.xml`

ConT_EXt ?

texexec.pl ?

Wege nach ...

Abbildungen

Beispiele

Schon bekannt

Close

4 Abbildungen

Breites Spektrum von Befehlen zur Darstellung von XML-Daten:

- „Darstellung von Elementen“
- „Darstellung leerer Elemente“
- „Behandlung von Attributen“
- „Darstellung von Entities“
- „Delimited Environments“
- „Pushing und Popping“

Eine umfangreichere Zusammenstellung der in diesem Zusammenhang möglichen XML-Befehlen finden Sie in **XML in ConT_EXt**.

ConT_EXt ?

texexec.pl ?

Wege nach ...

Abbildungen

Beispiele

Schon bekannt

Close

4.1 Darstellung von Elementen

Ausgangspunkt ist die folgende einfache XML-Konstruktion:

```
<element>inhalt</element>
```

Dann gibt es im Wesentlichen die folgenden Möglichkeiten:

a. `\defineXMLenvironment`:

```
\defineXMLenvironment [element] {\kommandoA} {\kommandoB}
```

⇒

```
\kommandoA inhalt \kommandoB
```

b. `\defineXMLcommand`:

```
\defineXMLcommand [element] {\kommando}
```

⇒

```
\kommando inhalt
```

c. `\defineXMLgrouped`:

```
\defineXMLgrouped [element] {\kommando}
```

⇒

```
{ \kommando inhalt }
```

ConTeXt ?

texexec.pl ?

Wege nach ...

Abbildungen

Beispiele

Schon bekannt

Close

d. `\defineXMLargument`:

```
\defineXMLargument [element] {\kommando}
```

\Rightarrow

```
\kommando { inhalt }
```

ConT_EXt ?

texexec.pl ?

Wege nach ...

Abbildungen

Beispiele

Schon bekannt

Close

4.2 Darstellung leerer Elemente

Leere XML-Elemente können gesondert behandelt werden:

- `\defineXMLsingular`:

```
\defineXMLsingular [element] {\kommando}
```

einsetzbar bei

```
<element />      <element></element>
```

- `\defineXMLcommand`:

```
\defineXMLcommand [element] {\kommando}
```

einsetzbar bei

```
<element></element>
```

Beide Formen führen zu

```
\kommando
```

ConT_EXt ?

texexec.pl ?

Wege nach ...

Abbildungen

Beispiele

Schon bekannt

Close

4.3 Behandlung von Attributen

Gegeben die folgende Konstruktion:

```
<element attribut="wert"> inhalt </element>
```

Auswahl einiger Befehle:

a. `\XMLpar`:

```
\XMLpar {element} {attribut} {voreinstellung}
```

liefert: *voreinstellung*, falls *wert* leer ist, anderenfalls *wert*

b. `\XMLifequalelse`:

```
\XMLifequalelse {element} {attribut} {abfrage}  
  {ifZweig} {elseZweig}
```

liefert: *ifZweig*, falls *abfrage* der Wert des Attributs *attribut* ist, anderenfalls *elseZweig*

c. `\mapXMLvalue`:

```
\mapXMLvalue {element} {attribut} {abfrage} {ersatz}
```

liefert: *ersatz*, falls *abfrage* der Wert des Attributs *attribut* ist

ConTeXt ?

texexec.pl ?

Wege nach ...

Abbildungen

Beispiele

Schon bekannt

Close

4.4 Darstellung von Entities

`<element> ... &entity; ... </element>`

- `\defineXMLentity:`
`\defineXMLentity [entity] {definition}`
 \Rightarrow
... definition ...

ConT_EXt ?

texexec.pl ?

Wege nach ...

Abbildungen

Beispiele

Schon bekannt

Close

4.5 Delimited Environments

- Aus HTML/XHTML bekannt:

```
<table>
<tr><td>...</td> ... <td>...</td></tr>
<tr><td>...</td> ... <td>...</td></tr>
...
</table>
```

- Gesucht Abbildung nach:

```
\bTABLE
\bTR \bTD...\eTD ... \bTD...\eTD \bTR
\bTR \bTD...\eTD ... \bTD...\eTD \bTR
...
\eTABLE
```

- Bewirkt wird diese Abbildung z.B. durch:

```
\defineXMLenvironment [table] {\bTABLE} {\eTABLE}
\defineXMLpickup      [tr]    {\bTR}      {\eTR}
\defineXMLpickup      [td]    {\bTD}      {\eTD}
```

- Bei geschachtelten Tabellen an Stelle von \defineXMLpickup:
\defineXMLnested

ConTeXt ?

texexec.pl ?

Wege nach ...

Abbildungen

Beispiele

Schon bekannt

Close

4.6 Pushing und Popping

Wenn die Elemente nicht in der *richtigen* Reihenfolge vorliegen,
... müssen sie zwischengespeichert werden:

```
...  
<figure label="kennung">  
  <caption>bildtitel</caption>  
  <graphic>datei</graphic>  
</figure>  
...
```

ConT_EXt ?

texexec.pl ?

Wege nach ...

Abbildungen

Beispiele

Schon bekannt

Close

Eine mögliche Abbildung:

```
\defineXMLenvironment [figure]
  {\bgroup
   \defineXMLpush[caption]
   \defineXMLpush[graphic]}
  {\placefigure
   []
   [fig:\XMLpar{figure}{label}{unknown}]
   {\XMLpop{caption}}
   {\externalfigure[\XMLpop{graphic}]}
  \egroup}
```

ConT_EXt ?

texexec.pl ?

Wege nach ...

Abbildungen


Beispiele

Schon bekannt

Close

5 Beispiele

Beispiel 1:

 6.xml: Abbildung mit \defineXMLenvironment, \defineXMLpickup

Direktivendatei tutc7.dir:

```
\def\textttt#1{{\tt #1}}
\def\adresse#1{\subsubject{Adresse~Nr.~#1}}
\def\startanrede#1\stopanrede{#1}
\def\starttitel#1\stoptitel{#1}
\def\startvorname#1\stopvorname{#1}
\def\startnachname#1\stopnachname{#1\par}
\def\startsname#1\stopsname{#1}
\def\starthnr#1\stopphnr{#1\par}
\def\startpostfach#1\stoppostfach{Postfach:~#1\par}
\def\startplz#1\stopplz{#1}
\def\startortsname#1\stoportsname{#1\par\smallskip}
\def\starttelefon#1\stoptelefon{Telefon:~#1\par}
\def\startfax#1\stopfax{Fax:~#1\par}
\def\startemail#1\stopemail{E-Mail:~\textttt{#1}}
```

ConT_EXt ?

texexec.pl ?

Wege nach XML

Abbildungen

Beispiele

Schon bekannt

Close

```

\defineXMLenvironment [adressen]
  {\useencoding[win]\starttext\title{Adressen}}
  {\stoptext}
\defineXMLpickup [adresse]
  {\adresse{\XMLpar{adresse}{nr}{--}}}
  {}
\defineXMLpickup [anrede]      \startanrede      \stopanrede
\defineXMLpickup [titel]       \starttitel       \stoptitel
\defineXMLpickup [vorname]     \startvorname     \stopvorname
\defineXMLpickup [nachname]    \startnachname    \stopnachname
\defineXMLpickup [sname]       \startsname       \stopsname
\defineXMLpickup [hnr]         \starthnr         \stophnr
\defineXMLpickup [postfach]    \startpostfach    \stoppostfach
\defineXMLpickup [plz]         \startplz         \stopplz
\defineXMLpickup [ortsname]    \startortsname    \stoportsname
\defineXMLpickup [telefon]     \starttelefon     \stoptelefon
\defineXMLpickup [fax]         \startfax         \stopfax
\defineXMLpickup [email]       \startemail       \stopemail

```

Aufruf: `texexec --pdf --env=tutc7.dir adr6.xml`

⇒ [Ergebnis](#)

[ConT_EXt ?](#)

[texexec.pl ?](#)

[Wege nach ...](#)


[Abbildungen](#)

[Beispiele](#)

[Schon bekannt](#)

[Close](#)

Beispiel 2:

 c8.tex: Abbildung mit `\defineXMLenvironment`, `\defineXMLargument`, `\XMLpar`, `\defineXMLgrouped`, `\defineXMLpush`, `\XMLpop`, Processing Instruction

```
% tutc8.tex
% adaptiert (nach Hans Hagen: XML in ConTeXT, example.pdf)

\defineXMLenvironment [text]  {\starttext} {\stoptext}
\defineXMLargument     [title]
    {\chapter[\XMLpar{title}{label}{}]}
\defineXMLgrouped      [tt]   {\tt}

\defineXMLenvironment [figure]
    {\bgroup
     \defineXMLpush[caption]
     \defineXMLpush[graphic]}
    {\placefigure
 %    [\XMLpar{figure}{location}{here}]
    []
     [fig:\XMLpar{figure}{label}{unknown}]}
```

ConTeXt ?

texexec.pl ?

Wege nach XML

Abbildungen

Beispiele

Schon bekannt

Close

```
{\XMLpop{caption}}
{\externalfigure[\XMLpop{graphic}]]}
\egroup}
```

```
%\useXMLfilter          [ent]
\defineXMLentity        [tex]  {\TeX}
```

```
\starttext
\startXMLdata
<text>
<title>Hello <tt>World</tt>, this is &tex;</title>
Welcome to this small example.
```

```
<figure label="figures">
  <caption>Some Caption</caption>
  <graphic>figdemo.pdf</graphic>
</figure>
```

```
<!-- Next some rather contextual code. The ? indicates
      that we are dealing with a processing instruction. -->
```

ConT_EXt ?

texexec.pl ?

Wege nach XML

Abbildungen

Beispiele

Schon bekannt

Close


```
<?context-command {\bf What a happy end to a small story!} ?>  
</text>  
\stopXMLdata  
\stoptext
```

Aufruf: texexec --pdf tutc8.tex

⇒ Ergebnis

ConT_EXt ?

texexec.pl ?

Wege nach ...

Abbildungen

Beispiele

Schon bekannt

Close

6 Schon bekannt

ConT_EXt kennt *von Haus aus* einige Dokumenttypen und interpretiert sie korrekt.

- **MATHML**: XML-Applikation zur Beschreibung mathematischer Formeln

- allgemeine Informationen:

- ★ [W3C's Math Home Page](#)
- ★ [Mathematical Markup Language \(MathML\) 1.01 Specification](#)

- Informationen (im Zusammenhang mit ConT_EXt):

- ★ [MATHML](#) (Übersicht)
- ★ [MathML](#) (Lehrbuch)
- ★ [MathML in ConT_EXt](#) (Beispiele)

ConT_EXt ?

texexec.pl ?

Wege nach ...

Abbildungen

Beispiele

Schon bekannt

Close

– Einsatz in ConT_EXt:

★ `\usemodule:`

...

`\usemodule[mathml]`

`\starttext`

`\startXMLdata`

...mathML

`\stopXMLdata`

★ `\useXMLfilter:`

...

`\useXMLfilter[mml,mmp,mmc]`

`\starttext`

`\startXMLdata`

...mathML

`\stopXMLdata`

★ Option `--usemodule` bzw. `--xmlfilter` beim Aufruf von `texexec`

– Testing MathML support in ConT_EXt

ConT_EXt ?

texexec.pl ?

Wege nach ...

Abbildungen

Beispiele

Schon bekannt

Close

– Beispiele:

- ★ „**Kleines Beispiel (MathML, useXMLfilter, ...**“
- ★ „**Beispiel (MathML, useXMLfilter, ...**“
- ★ „**Beispiel (MathML, useXMLfilter, ...**“

- **CHEMML**: XML-Applikation zur Beschreibung chemischer Formeln

- Informationen: **CHEMML**
- benötigtes Modul: `\usemodule[chemml]`

- **PHYSML**: korrekte und konsistente Schreibweise physikalischer Einheiten in Formeln

- Informationen: **PHYSML**
- Modul: `\usemodule[physml]`

- **FIGURES**: Datenbank zur Verwaltung von Abbildungen – realisiert als XML-Applikation

- Informationen: **FIGURES**
- benötigte Module: `\usemodule[fig-make]` und `\usemodule[fig-base]`

ConT_EXt ?

texexec.pl ?

Wege nach ...

Abbildungen

Beispiele

Schon bekannt

Close

- **STEPS**: XML-Applikation zur Beschreibung von Step-Charts
 - Informationen: **STEPS**
 - Modul: `\usemodule[steps]`
- **DocBook**: XML-Applikation zur Beschreibung (vorwiegend) technischer Dokumente
 - Informationen: **DocBook In ConTeXt – ConTeXt XML mapping for DocBook documents**
 - Autoren: Simon Pepping, Michael Wiedmann
 - Download:
<http://www.hobby.nl/~scaprea/context/DocbookInContext.tar.gz>
 - Einsatz: `\useXMLfilter[docbook]` oder entsprechende Option beim Aufruf von `texexec.pl`

ConTeXt ?

texexec.pl ?

Wege nach ...

Abbildungen

Beispiele

Schon bekannt

Zurück

Anfang des Tutoriums
Übersichtsvortrag

Close

Von XML nach PDF über T_EX – Internet-Adressen

Günter Partosch, Oktober 2002

Im Umfeld des Themenkreises *XML – PDF – T_EX* sind die folgenden Stellen im Internet von Interesse:

- alphaWorks: TeXML: Download
<http://www.alphaworks.ibm.com/aw.nsf/download/texml>
- alphaWorks: TeXML
<http://www.alphaworks.ibm.com/tech/texml>
- Apache XML Project: *FOP*
<http://xml.apache.org/fop>
- Berend de Boer (EuroT_EX'2001): *From database to presentation via XML, XSLT and ConTeXt*
<http://www.ntg.nl/eurotex/DeBoerSQL.pdf>
- David Carlisle: *XMLTEX: A non validating (and not 100% conforming) namespace aware XML parser implemented in TeX*
<ftp://ftp.dante.de/tex-archive/macros/xmltex/base/manual.html>
- David Carlisle: *xmltex package files*
<ftp://ftp.dante.de/tex-archive/macros/xmltex/base/manual.html#manualN651>

Close

- James Clark: *ISO/IEC 10179:1996. Document Style Semantics and Specification Language (DSSSL)*
<http://www.jclark.com/dsssl>
- Robin Cover: *The XML Cover Pages. SGML/XML and (La)TeX*
<http://xml.coverpages.org/sgml-tex.html>
- Diskussionsliste ntg-context@ntg.nl
<mailto:ntg-context@ntg.nl>
- DocBook Technical Committee: *DocBook*
<http://www.oasis-open.org/docbook/>
- Eitan M. Gurari, Sebastian Rahtz: *LaTeX to XML/MathML*
<http://www.cis.ohio-state.edu/~gurari/tug99/index.html>
- Hans Hagen: *ConT_EXt an excursion*
<http://www.pragma-ade.nl/general/manuals/mp-cb-en.pdf>
- Hans Hagen: *ConT_EXt*
<http://www.pragma-ade.nl/>
- Hans Hagen: *T_EXEXEC explained*
<http://www.pragma-ade.nl/general/manuals/mtexexec.pdf>

Close

- Hans Hagen: *T_EXUTIL explained*
<http://www.pragma-ade.nl/general/manuals/mtexutil.pdf>
- Hans Hagen: *XML in ConT_EXt*
<http://www.pragma-ade.nl/general/manuals/example.pdf>
- Hans Hagen: *CHEMML*
<http://www.pragma-ade.nl/general/manuals/xchemml-p.pdf>
- Hans Hagen: *FIGURES*
<http://www.pragma-ade.nl/general/manuals/xfigures-p.pdf>
- Hans Hagen: *how to install ConT_EXt*
<http://www.pragma-ade.nl/general/manuals/minstall.pdf>
- Hans Hagen: *MathML in ConT_EXt* (Beispiele)
<http://www.pragma-ade.nl/general/manuals/mmlexamp.pdf>
- Hans Hagen: *MathML* (Handbuch)
<http://www.pragma-ade.nl/general/manuals/mmlprime.pdf>
- Hans Hagen: *MATHML* (Überblick)
<http://www.pragma-ade.nl/general/manuals/xmathml-p.pdf>

Close

- Hans Hagen: *PHYSML*
<http://www.pragma-ade.nl/general/manuals/xphysml-p.pdf>
- Hans Hagen: *Read Me First*
<http://www.pragma-ade.nl/general/manuals/mreadme.pdf>
- Hans Hagen: *STEPS*
<http://www.pragma-ade.nl/general/manuals/xsteps-p.pdf>
- Hans Hagen: *The ConTeXt XML Page*
<http://www.pragma-ade.nl/xml.htm>
- Hans-Hagen: offizielle ConT_EXt-Distribution
<http://www.pragma-ade.nl/context/cont-tfm.zip>
- Hans Hagen, Kees van Marle, Ton Otten: *Testing MathML support in ConTeXt*
<http://www.pragma-pod.nl/podindex.htm>
- Simon Pepping (EuroT_EX'2001): *From XML to TeX: an overview of available methods*
<http://www.ntg.nl/eurotex/PeppingXML.pdf>

Close

- Simon Pepping, Michael Wiedmann: *DocBook In ConTeXt – ConTeXt XML mapping for DocBook documents*
<http://www.hobby.nl/~scaprea/context/>
- Simon Pepping, Michael Wiedmann: DocBook In ConTeXt (Download)
<http://www.hobby.nl/~scaprea/context/DocbookInContext.tar.gz>
- Sebastian Rahtz (DANTE'2001): *Future of TeX in a XML-dominated world*
<http://www.dante.de/dante2001/handouts/rahtz-xml/rahtz-xml.ps>
- Sebastian Rahtz: PassiveTeX-Download
<http://www.tei-c.org.uk/Software/passivetex/passivetex.zip>
- Sebastian Rahtz: *Text Encoding Initiative: XSL stylesheets for TEI XML*
<http://www.tei-c.org.uk/Stylesheets/teixsl.html>
- Sebastian Rahtz: *Text Encoding Initiative – PassiveTeX*
<http://www.tei-c.org.uk/Software/passivetex/>
- Sebastian Rahtz: *JadeTeX*
<http://jadetex.sourceforge.net>

Close

- SourceForge: *Project Info – jadetex*
<http://sourceforge.net/projects/jadetex>
- Text Encoding Initiative: *Welcome to the TEI Website*
<http://www.tei-c.org/>
- W3C: *CSS1*
<http://www.w3.org/TR/REC-CSS1>
- W3C: *CSS2*
<http://www.w3.org/TR/REC-CSS2/>
- W3C: *CSS3 introduction*
<http://www.w3.org/TR/css3-roadmap>
- W3C: *Extensible Markup Language (XML) 1.0 (Second Edition)*
<http://www.w3.org/TR/REC-xml>
- W3C: *Extensible Stylesheet Language (XSL) Version 1.0*
<http://www.w3.org/TR/xsl>
- W3C: *Mathematical Markup Language (MathML) 1.01 Specification*
<http://www.w3.org/TR/REC-MathML>

Close

- W3C: *Namespaces in XML*
<http://www.w3.org/TR/REC-xml-names>
- W3C: *W3C's Math Home Page*
<http://www.w3.org/Math/>
- W3C: *XHTML 1.0: The Extensible HyperText Markup Language - A Reformulation of HTML 4 in XML 1.0*
<http://www.w3.org/TR/xhtml1>
- W3C: *XML Information Set*
<http://www.w3.org/TR/xml-infoset>
- W3C: *XSL Transformations (XSLT) Version 1.0*
<http://www.w3.org/TR/xslt>

Close

Unser Kapitel

Emil Mayer

28. Februar 2002

1 Titel des ersten Kapitels

Ein bisschen Text ...

 Noch ein bisschen Text ...

2 Titel des zweiten Kapitels

Ein bisschen anderer Text ...

 Noch ein bisschen anderer Text ...

Unser Kapitel

Emil Mayer

28. Februar 2002

1 Titel des ersten Kapitels

Ein bisschen Text ...

 Noch ein bisschen Text ...

2 Titel des zweiten Kapitels

Ein bisschen anderer Text ...

 Noch ein bisschen anderer Text ...

Adressen

Emil Mayer

14. Februar 2002

Adressen

Adresse~Nr.~1

Frau Dr. Anna Mayer
Blumenweg 2
35392 Gießen

Telefon:~0641-123456

Fax:~0641-123459

E-Mail:~Anna.Mayer@t-online.de

Adresse~Nr.~2

Herr Emil Müller
Tulpenweg 22a
35398 Gießen

Telefon:~0641-234567

E-Mail:~Emil.Mueller@profanistik.uni-giessen.de

Adresse~Nr.~3

Frau Prof. Dr. Erna Schneider
Bachstraße 10
Postfach:~12345
12340 Nixdorf

Telefon:~09994-1234

Fax:~09994-12349

E-Mail:~Erna.Schneider@gmx.de

xmltex: A non validating (and not 100% conforming) namespace aware XML parser implemented in T_EX

David Carlisle

Date: 2000-01-24

1 Introduction

xmltex implements a non validating parser for documents matching the W3C XML Namespaces Recommendation. The system may just be used to parse the file (expanding entity references and normalising namespace declarations) in which case it records a trace of the parse on the terminal. Normally however the information from the parse is used to trigger T_EX typesetting code. Declarations (in T_EX syntax) are provided as part of xmltex to associate T_EX code with the start and end of each XML element, attributes, processing instructions, and with unicode character data.

2 Installation

The xmltex parser itself does not require L^AT_EX. It may be loaded into initex to produce a format capable of parsing XML files. However such a format would have no convenient commands for typesetting, and so normally xmltex will be used on top of an existing format, normally L^AT_EX. In this section we assume that the document to be processed is called `document.xml`.

2.1 Using xmltex as an input to the L^AT_EX command

L^AT_EX requires a document in T_EX syntax, not XML. To process `document.xml`, first produce a two line file called `document.tex` of the following form:

```
\def\xmlfile{document.xml}
\input xmltex.tex
```

Do *not* put any other commands in this file!

You may then process the document with either of the commands: `latex document` or `latex document.tex` or the equivalent procedure in your T_EX environment.

2.2 Using `xmltex` as a \TeX format built on \LaTeX

You may prefer to set up `xmltex` as a format in its own right. This may speed things up slightly (as `xmltex.tex` does not have to be read each time) but more importantly perhaps it allows the XML file to be processed directly without needing to make the `.tex` wrapper.

To make a format you will need a command such as the following, depending on your \TeX system.

```
initex &latex xmltex
initex \&latex xmltex
tex -ini &latex xmltex
tex -ini \&latex xmltex
```

This will produce a format file `xmltex.fmt`. You should then be able to make a `xmltex` command by copying the way the `latex` command is defined in terms of `latex.fmt`. Depending on the \TeX system, this might be a symbolic link, or a shell script, or batch file, or a configuration option in a setup menu.

2.3 Making an `xmltex` format ‘from scratch’

Whilst it may be convenient to build an `xmltex` format as above, starting from the \LaTeX format. You may prefer to instead work with an `initex` with no existing format file. Even if you wish to use a standard \LaTeX it may be preferable to make a \TeX input file that first inputs `latex.ltx` then `xmltex.tex`. In particular this will allow you to have a different hyphenation and language customisation for `xmltex` than for \LaTeX . Many of the features of the language support in \LaTeX are related to modifying the input syntax to be more convenient. Such changes are not needed in `xmltex` as the input syntax is always XML. Some language files may change the meaning of such characters as `<` which would break the `xmltex` parser. Also, rather than using `latex.ltx` you could in principle use a modified `docstrip` install file and produce a ‘cut down’ `latex` that did not have features that are not going to be used in `xmltex`.

Unfortunately the support for this method of building `xmltex` (and access to non English hyphenation generally) is not fully designed and totally undocumented.

3 Using `xmltex`

`xmltex` by default ‘knows’ nothing about any particular type of XML file, and so needs to load external files containing specific information. This section describes how the information in the XML file determines which files will be loaded.

1. If the file begins with a Byte Order Mark, the default encoding is set to utf-16. Otherwise the default encoding is utf-8.

2. If (after an optional BOM) the document begins with an XML declaration that specifies an encoding, this encoding will be used, otherwise the default encoding will be used. A file with name of the form *encoding.xmlt* will be loaded that maps the requested encoding to Unicode positions. (It is an error if this file does not exist for the requested encoding.)
3. If the document has a DOCTYPE declaration that includes a local subset then this will be parsed. If any external DTD entity is referenced (by declaring and then referencing a parameter entity) then the SYSTEM and PUBLIC identifiers of this entity will be looked up in a catalogue (to be described below). If either identifier is known in the catalogue the corresponding xmltex package (often with .xmlt extension) will be loaded.
4. After any local subset has been processed, if the DOCTYPE specifies an external entity, the PUBLIC and/or SYSTEM identifiers of the external dtd file will be similarly looked up, and a corresponding xmltex file loaded if known.
5. As each element is processed, it may be ‘known’ to xmltex by virtue of one of the packages loaded, or it may be unknown. If it is unknown then if it is in a declared namespace, the namespace URI (not the prefix) is looked up in the xmltex catalogue. If the catalogue specifies an xmltex package for this namespace it will be loaded. If the element is not in a namespace, then the element name will be looked up in the catalogue.
6. If after all these steps the element is still unknown then depending on the configuration setting either a warning or an error will be displayed. (Currently only warning implemented.)

3.1 The xmltex Catalogue

As discussed above, xmltex requires a mapping between PUBLIC and SYSTEM identifiers, namespace URI, and element names, to files of T_EX code. This mapping is implemented by the following commands:

```

\NAMESPACE{URI}{xmt-file}
\PUBLIC{FPI}{file}
\SYSTEM{URI}{file}
\NAME{element-name}{xmt-file}
\XMLNS{element-name}{URI}

```

As described above, if the first argument of one of these commands matches the string specified in the XML source file, the corresponding T_EX commands in the file specified in the second argument are loaded. The PUBLIC and SYSTEM catalogue entries may also be used to control which XML files should be input in response to external entity references. The \XMLNS is rather different, if an element in the null

namespace does not have any definition attached to it, this declaration forces the default namespace to the given URI. The catalogue lookup is then repeated. This allows for example documents beginning `<html>` to be coerced into the `xhtml` namespace.

These commands may be placed in a configuration file, either `xmlltex.cfg`, in which case they apply to all documents, or in a configuration file `'\jobname.cfg'` (eg `document.cfg` in the example in the Introduction) in which case the commands just apply to the specified document.

3.2 Configuring xmlltex

In addition to the ‘catalogue’ commands described earlier there are other commands that may be placed in the configuration files.

- `\xmltraceonly`

This stops xml from trying to typeset the document. The external files specified in the catalogue are still loaded, so that the trace may report any elements for which no code is defined, but no actual typesetting takes place. In the event of unknown errors it is always worth using xmlltex in this mode to isolate any problems.

It may be noted that if an xmlltex format is built just using initex without any typesetting commands, the resulting format should still be able to parse any XML file if `xmlltex.cfg` just specifies `\xmltraceonly` and `\jobname.cfg` is empty.

- `\xmltraceoff`

By default xmlltex provides a trace of its XML parse, displaying each element begin and end. This command used in `xmlltex.cfg` or `'\jobname.cfg'` will stop this trace being produced.

- `\inputonce{xmt-file}`

The catalogue entries specify that certain files should be loaded if XML constructs are met. Alternatively the files may just always be loaded. The system will ignore any later requests to load. This is especially useful if an xmlltex format is being made.

- `\UnicodeCharacter{hex-or-dec}{tex-code}`

The first argument specifies a unicode character number, in the same format as used for XML character entities, namely either a decimal number, or an upper case Hex number preceded by a lower case ‘x’.

The second argument specifies arbitrary \TeX code to be used when typesetting this character. Any code in the XML range may be specified (ie up to `x10FFFF`). Although codes in the ‘ASCII’ range, below 128, may be specified, the definitions supplied for such characters will not be default be used. The definition will however be stored and used if the character is activated using the command described below.

- `\ActivateASCII{hex-or-dec}`

The argument to this command should be a number less than 128. If a character is activated by this command in a configuration file then any special typesetting instructions specified for the character will be executed whenever the character appears as character data.

Some ASCII characters are activated by default. The list is essentially those characters with special meanings to either \TeX or XML.

If a format is being made, there are essentially two copies of `xmlltex.cfg` that may play a role. The configuration file input when the format is made will control catalogue entries and packages built into the format. A possibly different `xmlltex.cfg` may be used in the input path of ‘normal’ \TeX , this will then be used for additional information loaded each run.

In either case, a separate configuration file specific to the given XML document may also be used (which is loaded immediately after `xmlltex.cfg`).

4 Stopping xmlltex

In tracing mode, xmlltex will stop after the end of the document element has been processed. In the normal processing mode xmlltex does not currently automatically stop \TeX processing. After the document has completed \TeX will move to the interactive * prompt (from which you might want to exit with `<?xmlltex stop?>`). Normally the end code specified for the document element should execute `\end{document}` and so stop processing and avoid the * prompt.

5 xmlltex package files

xmlltex package files are the link between the XML markup and \TeX typesetting code. They are written in \TeX (rather than XML) syntax and may load directly or indirectly other files, including \LaTeX class and package files. For example a file loaded for a particular document type may directly execute `\LoadClass{article}`, or alternatively it may cause some XML element in the document to execute `\documentclass{article}`. In either case the document will suffer the dubious benefit of being formatted based on the style implemented in `article.cls`. Beware though that the package files may be loaded at strange times, the first time a given namespace is declared in a document, and so the code should be written to work if loaded inside a local group.

Characters in xmlltex package files have their normal \LaTeX meanings except that line endings are ignored so that you do not need to add a % to the end of lines in macro code. Unlike fd file conventions, other white space is *not* ignored.

The available commands are:

- `\FileEncoding{encoding}`

This is the analogue for \TeX syntax files of the encoding specification in the XML or text declaration of XML files. If it is not specified the file will be assumed to be in UTF-8.

- `\DeclareNamespace{prefix}{URI}`

This declares a prefix to be used *in this file* for referring to elements in the specified namespace. If the prefix is empty then this declares the default namespace (otherwise, unprefixed element names refer to elements that are not in a namespace).

Note that the elements in the XML document instance may use a different prefix, or no prefix at all to access this namespace. In order to resolve these different prefixes for the same namespace, each time a namespace is encountered for the first time (either by `\DeclareNamespace` in a preloaded package, or in a namespace declaration in the XML instance) then it is allocated a new number and any further namespace declaration for the same URI just locally associates a prefix with this number. It is these numbers that are displayed when the XML trace of the parse of the document is shown, and also if any element is written out to an external file it will have a normalised prefix of a number whichever prefix it had originally. (Numeric prefixes are not legal XML, but this is an advantage, it ensures these internal forms can not clash with any prefix actually used in the document.)

Three namespaces are predeclared. The null namespace (0), the XML namespace (<http://www.w3.org/1998/xml>) (1) which is predeclared with prefix `xml` as specified in the Namespace Recommendation, and the `xmltex` namespace (<http://www.dcarlisle.demon.co.uk/xmltex>) (2) which is not given a default prefix, but may be used to have XML syntax for some internal commands (eg to have `.aux` files fully in XML, currently they are a hybrid mixture of some \TeX and some XML syntax).

- `\XMLelement{element-qname}{attribute-spec}{begin-code}{end-code}`

This is similar to a \LaTeX `\newenvironment` command.

Declare the code to execute at the start and end of each instance of this element type. This code will be executed in a local group (like a \LaTeX environment). The second argument declares a list of attributes and their default values using the `\XMLattribute` command described below.

- `\XMLelement{element-qname}{attribute-spec}{\xmlgrab}{end-code}`

A special case of the above command (which may be better made into a separate declaration) is to make the *start-code* just be the command `\xmlgrab`. In this case the *end-code* has access to the element content (in XML syntax) as #1. This content isn't literally the same as the original document, namespaces, white space and attribute quote symbols will all have been normalised.

- `\XMLattribute{attribute-qname}{command-name}{default}`

This command may only be used in the argument to `\XMLelement`. The first argument specifies the name of an attribute (using any namespace prefixes current for this package file, which need not be the same as the prefixes used in the document). The second argument gives a \TeX command name that will be used to access the value of this attribute in the begin and end code for the element. (Note using \TeX syntax here provides a name independent of the namespace declarations that are in scope when this code is executed). The third argument provides a default value that will be used if the attribute is not used on an instance of this element.

The special token `\inherit` may be used which will cause the command to have a value set in an ancestor element if this element does not specify any value.

If a \TeX token such as `\relax` is used as the default the element code may distinguish the case that the attribute is not used in the document.

- `\XMLnamespaceattribute {prefix}{attribute-qname} {command-name} {default}`

This command is similar to `\XMLattribute` but is used at the top level of the package file, not in the argument to `\XMLelement`. It is equivalent to specifying the attribute in *every* element in the namespace specified by the first argument. As usual the prefix (which may be `{ }` to denote the default namespace) refers to the namespace declarations in the `xmltex` package: the prefixes used in the document may be different.

- `\XMLentity{name}{code}`

Declare an (internal parsed) entity, this is equivalent to a `<!ENTITY` declaration, except that the replacement text is specified in \TeX syntax.

- `\XMLname{name}{command-name}`

Declare the \TeX command to hold the (normalised, internal form) of the XML name given in the first argument. This allows the code specified in `\XMLelement` to refer to XML element names without knowing the encodings or namespace prefixes used in the document. Of particular use might be to compare such a name with `\ifx\xml@parent` which will allow element code to take different actions depending on the parent of the current element.

- `\XMLstring{command-name}<>XML Data</>`

This saves the XML fragment as the \TeX command given in the first argument. It may be particularly useful for redefining ‘fixed strings’ that are generated by \LaTeX document classes to use any special typesetting rules specified for individual characters.

6 XML processing

`xmltex` tries as far as possible to be a fully conforming non validating parser. It fails in the following respects.

- Error reporting is virtually non-existent. Names are not checked against the list of allowed characters, and various other constraints are not enforced.
- A non-validating parser is not forced to read external dtd entities (and this one does not). It is obliged to read the local subset and process entity definitions and attribute declarations. Entity declarations are reasonably well handled: External parameter entities are handled as above, loading a corresponding `xmltex` file if known. External entities are similarly processed, inputting the XML file, a difference in this case is that if the entity is not found in the catalogue, the `SYSTEM` identifier will be used directly to `\input` as often this is a local file reference. Internal parsed entities and parameter entities are essentially treated as `TEX` macros, and nonparsed entities are saved along with their `NDATA` type, for use presumably by `\includegraphics`.

What is not currently done, but mandated by the XML 1.0 Recommendation is that default attributes are read from the internal subset. (This might be added later.)

- Support for encodings depends on having an encoding mapping file. Any 8bit encoding that matches Unicode for the first 127 positions may be used by making a trivial mapping file. (The one for latin1 looks over complicated as it programs a loop rather than having 127 declarations saying that latin1 and Unicode are identical in this range).

UTF-8 is supported, but support for UTF-16 is minimal. Currently only latin-1 values work: (In this range UTF-16 is just latin-1 with a null byte inserted after (or before, depending on endedness) each latin-1 byte. The UTF-16 implementation just ignores this null byte then processes as for latin-1. Probably the first few 8bit pages could be similarly supported by making the low ascii control characters activate UTF-16 processing but this will never be satisfactory using a standard `TEX`. Hopefully a setup for a 16bit `TEX` such as Omega will correct this.

7 Accessing `TEX`

In theory you should be able to control the document just by suitable code specified by `\XMLelement` and friends, but sometimes it may be necessary to ‘tweak’ the output by placing commands directly in the source.

Two mechanisms are available to do this.

- Using the `xmltex` namespace. The `xmltex` namespace contains a small (currently empty) set of useful `TEX` constructs that are accessed by XML syntax. For example if `xmltex` provides a mechanism for having XML (rather than `LATEX`) syntax toc files, it will need an analogue of `\contentsline` which might be an element accessed by `<xmltex:contentsline>...` where the `xmltex` prefix is declared on this or a parent element to be `xmlns:xmltex="http://www.dcarlisle.demon.co.uk/xmltex"`.

As the `xmltex` namespace is declared but currently empty, a more useful variant of this might be:

- Declare your own namespace for T_EX tweaks, and load a suitable package file that attaches T_EX code to the elements in this namespace (or at least specify the correspondence between the namespace and the package using `\NAMESPACE`). For instance if you put `<clearpage xmlns="/my/tex/tweak"/>` in your document, this will force a page break if you have at suitable points, `\NAMESPACE{/my/tex/tweak}{tweak.xmt}` and

```
\DeclareNamespace{tweak}{"/my/tex/tweak"}
\XMLelement{tweak:clearpage}{\clearpage}
```

- A second different mechanism is available, to use XML processing instructions. A Processing Instruction of the form: `<?xmltex> name {arg1} {arg1} ... ?>` will be executed as the T_EX command: `\name {arg1} {arg1} ...`

Note that *only* the first name will be converted into a T_EX command, any arguments should only contain character strings.

8 Bugs

None, of course.

9 Don't Read Past This Point

This section discusses some of the more experimental features of `xmltex` that may get a cleaner syntax (or be removed, as a bad idea) in later releases, and also describes some of the internal interfaces (which are also subject to change)

9.1 Input Encodings and States

At any point while processing a document, `xmltex` is in one of two *states*: *tex* or *xml*.

9.1.1 States

In the *xml state*, `<` and `&` are the only two characters that trigger special markup codes. Other characters, such as `!`, `>`, `=`, ... may be used in certain XML constructs as markup but unless some code has been triggered by `<` they are treated simply as character data. All characters above 127 are 'active' to T_EX and are used to translate the input encoding to UTF-8. All internal character handling is based on UTF-8, as described below. Some characters in the ASCII range, below 127 are also active by default (mainly punctuation characters used in XML constructs, such as the ones listed above). Some or all of the others may be activated using the `\ActivateASCII` command, which allows special typesetting rules to be activated for the characters, at some cost in processing speed.

In the *tex state*, characters in the ASCII range have their usual T_EX meanings, so letters are 'catcode 11' and may be used in T_EX control sequences, `\` is the escape

character, & the table cell separator, etc. Characters above 127 have the meanings current for the current encoding just as for the xml state, probably this means that they are unusable in T_EX code, except for the special case of referring to XML element names in the first argument to \XMLelement and related commands.

9.1.2 Encodings

Whenever a new (XML or T_EX) file is input by the xmltex system the *encoding* is first switched to UTF-8. At the end of the input the encoding is returned to whatever was the current encoding. The encoding current while the file is read is determined by the encoding pseudo-attribute on the XML or text declaration in the case of XML files, or by the \FileEncoding command for T_EX files. Note that the encoding mechanism *only* is triggered by xmltex file includes. Once an xmltex package file is loaded it may include other T_EX files by \input or \includepackage these input command will be transparent to the xmltex encoding system. The vast majority of T_EX macro packages only use ASCII characters so this should not be a problem.

Note that if the \includepackage occurs directly in the xmltex package file, the T_EX code will be included with a known encoding, the one specified in the xmltex package, or UTF-8. If however the \includepackage is included in code specified by \XMLelement, then it will be executed with whatever encoding is current in the document at the point that element is reached. Before xmltex executes the code for that element it will switch to the tex state, thus normalising the ascii characters but characters above 127 will not have predefined definitions in this case.

Internally everything is stored as UTF-8. So ‘aux’ and ‘toc’ files will be in UTF-8 even if the document (or parts of the document) used different encodings.

To specify a new encoding, if it is an 8 bit encoding that matches ASCII in the printable ASCII range, then one just needs to produce a file with name *encoding.xmt* (in lowercase, on case sensitive systems) this should consist of a series of \InputCharacter commands, giving the input character slot and the equivalent Unicode. If an encoding is specified in this manner character data will be converted to UTF-8 by *expansion* and so ligatures and inter letter kerns will be preserved. (Conversely if characters are accessed by character references, Ӓ then T_EX arithmetic is used to decode the information and ligature information will be lost. For some large character sets, especially for Asian languages, these mechanisms will probably not prove to be sufficient, some mechanisms are being investigated, but in the short term it may be necessary to always use UTF-8 if the input encoding is not strictly a one byte extension of the ASCII code page.

9.2 xmltex Package Commands

You can use arbitrary T_EX commands in an xmltex package, although you should be aware that the file may be input into a local group, at the point in a document that a particular namespace is first used, for example. There are however some specific commands designed to be used in the begin or end code of \XMLelement.

- \ignorespaces

This is actually a \TeX primitive (for the moment!)

- `\obeyspaces`

Obey consecutive space characters, rather than treating consecutive runs as a single space. (A command of this name, but not this definition is in plain \TeX .)

- `\obeylines`

Obey end of line characters, rather than treating them as a space, force a line break. (A command of this name, but not this definition is in plain \TeX .)

- `\xmltexfirstchild#1\@`

If the *start-code* for an element is specified as `\xmlgrab` then the *end-code* may use `#1` in order to execute the element content. Sometimes you do not want all of the content. The a construction (with currently unpleasant syntax) `\xmltexfirstchild#1\@` will just evaluate the first child element of the content, discarding the remaining elements.

- `\xmltextwochildren\csa\csb#1`

If you know that the content will be exactly two child elements (for example a MathML `frac` or `sub` element) then this command may be used. It will execute the \TeX code `\csa{child-1}\csb{child-2}`. So either two \TeX commands may be supplied, one will be applied to each child, or the second argument may be `{}` in which case the first argument may be a \TeX command that takes two arguments. For example the code for MathML `frac` might be

```
\XMLelement{m:mfrac}
{
  {\xmlgrab}
  {\xmltextwochildren\frac{}}{#1}}

```

- `\xmltextthreechildren\csa\csb\csc#1`

As above, but more so.

- `\NDATAEntity\csa\csb\attvalue`

If the XML parser encounters an internal or external entity reference it expands it without executing any special hook that may be defined in an `xmltex` package. However `NDATA` entities are never directly encountered in an entity reference. They may only be used as an attribute value. If `\attvalue` is a \TeX command holding the value of an attribute, as declared in `\XMLattribute` then `\NDATAEntity\csa\csb\attvalue` applies the two \TeX commands `\csa` and `\csb` to the notation type and the value, in a way exactly corresponding to `\xmltextwochildren` so for example the XML document for this manual specifies

```

<!NOTATION URL SYSTEM "" >
<!ENTITY lppl SYSTEM "http://www.latex-project.org/lppl.txt"
NDATA URL>

```

and this is handled by the following xmltex code

```

\XMLelement{xptr}
{\XMLattribute{doc}{\xptrdoc}{}}
{\NDATAEntity\xptrdoc\@gobble\url}
{}

```

which saves the attribute value in `\xptrdoc` and then discards the notation name (URL) and applies the command `\url` to typeset the supplied URL.

9.3 Character Data Internals

	int.	ext. xml	ext. mixed	csn typeout		
d	xabc	xabc	xabc (12)	xabc (12)	xabc (12)	
c	xab	xab	xab (12)	xab (12)	xab (12)	
b	xa	xa	xa (12)	xa (12)	xa (12)	
ax	x	x	x	x	x (12)	(!)
ay	x	x	x	{	x (12)	(e)
az	x	\az x	{	{	x (12)	(<)
<	<	<	<	<	< (12)	(<)

normaler Text – **fetter Text** – normaler Text – normaler Text – normaler Text –
normaler Text – normaler Text – normaler Text – normaler Text – normaler Text –
normaler Text –
normaler Text – mit kürzerem *XML-Einsprengsel* – normaler Text – normaler Text
– **fetter Text** – normaler Text – normaler Text – normaler Text – normaler Text –
normaler Text – normaler Text –
normaler Text – normaler Text – normaler Text – normaler Text – normaler Text –
normaler Text – normaler Text – normaler Text – normaler Text – normaler Text –
normaler Text –

XML-Einsprengsel

```
<math>
  <apply><eq />
    <apply><power /> <ci>a</ci> <ci>2</ci> </apply>
    <apply><plus />
      <apply><power /> <ci>b</ci> <ci>2</ci> </apply>
      <apply><power /> <ci>c</ci> <ci>2</ci> </apply>
    </apply>
  </apply>
</math>
```

$$a^2 = b^2 + c^2$$

$$1$$

$$\sin(a+b)$$

$$1$$

$$a^2 = b^2 + c^2$$

$$a + b + c$$

1 Titel des ersten Kapitels

Ein bisschen Text ...

Noch ein bisschen Text ...

Adressen

Adresse Nr. 1

Frau Dr. Anna Mayer
Blumenweg 2
35392 Gießen
Telefon: 0641-123456
Fax: 0641-123459
E-Mail: `Anna.Mayer@t-online.de`

Adresse Nr. 2

Herr Emil Müller
Tulpenweg 22a
35398 Gießen
Telefon: 0641-234567
E-Mail: `Emil.Mueller@profanistik.uni-giessen.de`

Adresse Nr. 3

Frau Prof. Dr. Erna Schneider
Bachstraße 10
Postfach: 12345
12340 Nixdorf
Telefon: 09994-1234
Fax: 09994-12349
E-Mail: `Erna.Schneider@gmx.de`

1 Hello World, this is T_EX

Welcome to this small example.

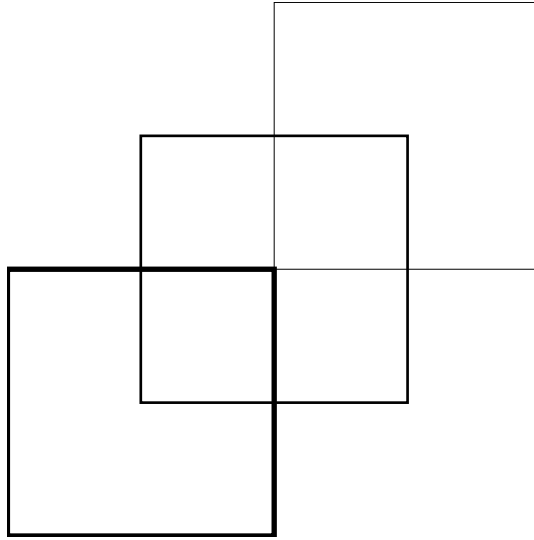


Figure 1.1 Some Caption

What a happy end to a small story!