XSLT – eXtensible Style Language Transformation

XSLT ist eine Transformationssprache für XML-Dokumente. Ein XSLT-Prozessor verarbeitet ein XML-Dokument durch die Angaben in einem XSLT-Dokument und erzeugt ein Ergebnisdokument (oder mehrere) in einem Format (z.B. XML, HTML), das vom XSLT-Dokument bestimmt wird.

Beispiel

```
❶ <?xml version="1.0" encoding="UTF-8"?>
❷ <xsl:stylesheet xmlns:xsl="http://www.w3.org/
  1999/XSL/Transform" version="1.0">
❸   <xsl:template match="/">
❹     Vorkommende Elementnamen:<br/><ul>
❺       <xsl:for-each select="//*">
❻         <li><xsl:value-of select="name()"/></li>
        </xsl:for-each></ul>
❼     <xsl:apply-templates/>
    </xsl:template>
❽   <xsl:template match="titel">
❾     <h1>Titel: <xsl:value-of select="."/></h1>
    </xsl:template>
  </xsl:stylesheet>
```

❶ Ein XSLT-Stylesheet ist ein XML-Dokument!
❷ Wurzelelement xsl:stylesheet; hier XSLT 1.0
❸ Eine allgemeine Schablone trifft auf die Wurzel (“/”) des XML-Dokuments zu
❹ In das Ergebnisdokument wird etwas geschrieben (hier: ein String, ein Umbruch, das öffnende Tag für eine Liste (HTML))
❺ Für jedes Element in einem Knotensatz (hier: “//*” = beliebige Elemente in beliebiger Tiefe) soll etwas getan werden
❻ Es wird etwas in das Ergebnisdokument geschrieben. Dazwischen wird ein Wert ausgegeben (hier: der Name des jeweils aktuellen Knotens)
❼ Berücksichtige ggf. weitere, in diesem Kontext passende Schablonen
❽ Eine Schablone, die auf Element `titel` anzuwenden ist
❾ Es wird etwas in das Ergebnisdokument geschrieben. Dazwischen wird ein Wert ausgegeben (hier: der (Text-)Inhalt des aktuellen Knotens)

→ Dieses Beispiel-XSLT-Sheet produziert eine Liste mit Elementnamen und gibt Überschriften aus

Weitere Referenzen

http://www.w3schools.com/XSL/xsl_w3celementref.asp (kurz)
http://de.selfhtml.org/xml/darstellung (deutsch)
http://www.zvon.org/xxl/XSLTreference/Output/index.html (ausführlicher, mit Beispielen)

Wichtige XSLT-Elemente (von insgesamt ca. 50)

`<xsl:template match="XPath-Ausdruck" name="String">`
– Eine Schablone, ggf. mit Namen; darin ggf. Parameterannahme mit `<xsl:param name="String"/>`

`<xsl:apply-templates select="XPath"/>`
– berücksichtige in diesem Kontext weitere Schablonen (ggf. für bestimmte Elemente)

`<xsl:call-template name="String">`
– ruft eine andere Schablone mit ihrem Namen auf; darin ggf. Parameterübergabe mit `<xsl:with-param name="String" select="XPath"/>`

`<xsl:value-of select="XPath"/>`
– hole etwas aus dem XML-Dokument und schreibe es in das Ergebnisdokument. Kurzsyntax innerhalb von Attributkonstruktionen: *attribut*="{XPath}"

`<xsl:for-each select="XPath"/>`
– tue etwas für jedes Element einer Knotenmenge

`<xsl:sort select="XPath" data-type="text | number" order="ascending | descending"/>`
– Leeres Element! Unmittelbar nach for-each! Sortiert eine Knotenmenge.

`<xsl:if test="Ausdruck">`
– Nur unter einer Bedingung …

`<xsl:choose>`
– wählt unter verschiedenen Bedingungen, die von den Kindelementen `<xsl:when test="Ausdruck">`+ und `<xsl:otherwise>`? beschrieben werden.

`<xsl:variable name="String">`
– eine Variable; der Variableninhalt wird innerhalb des Elements aufgebaut

`<xsl:result-document href="String">`
– Erlaubt die Generierung mehrerer Ergebnisdokumente. Deren Dateiname kann in String konstruiert werden. XSLT 2.0 !

`<xsl:element name="String">`
– konstruiert ein Element

`<xsl:attribute name="String">`
– konstruiert ein Attribut

`<xsl:text>`
– schreibt Zeichendaten in das Ergebnisdokument

`<xsl:copy>`
– erzeugt eine “flache” Kopie (keine Kinder, keine Attribute) des Kontextknotens

`<xsl:copy-of select="XPath">`
– erzeugt eine “tiefe” Kopie (mit allen Nachkommen)

`<xsl:output method="xml | html | …" …/>`
– (verschiedene weitere Attribute) gibt an, wie das Ergebnisdokument ausgegeben werden soll

Häufige Fehlerquellen

Der aktuelle Kontextknoten bei der Verarbeitung ist ein anderer, als man vermutet! Deshalb wird nichts ausgegeben.
Es gibt vordefinierte templates: Textinhalte werden immer ausgegeben (außer man unterdrückt das ausdrücklich: `<xsl:template match="text()"/>`)
Ein Namespace ist referenziert, wird aber in den XPath-Ausdrücken nicht berücksichtigt (keine Ausgabe!).

i|d|e
www.i-d-e.de

Historisch Kulturwissenschaftliche Informationsverarbeitung
www.hki.uni-koeln.de

XML

Kurzreferenz für Einsteiger

(mit DTD, XPath und XSLT)

Patrick Sahle

XML – eXtensible Markup Language

Beispiel

```
❶ <?xml version="1.0" encoding="UTF-8"?>
❷ <!DOCTYPE gedicht SYSTEM "Dateiname.dtd">
❸ <?xml-stylesheet type="text/xsl"
                   ❺      href="Dateiname.xsl"?>
❹ <gedicht xmlns= "URI">
              ❼   ❽          ❾
❻    <titel lang="de"> Kreti &amp; Pleti</titel>
                    ⓫
❿    <strophe>...<br/>...</strophe>
⓬    <!-- Kommentar -->
  </gedicht>
```

❶ XML-Deklaration
❷ Dokumenttyp-Deklaration (DTD-Verknüpfung) (optional)
❸ Processing-Instruction (Stylesheet-Verknüpfung) (optional)
❹ Element (`Name= "gedicht"`), Wurzelelement
❺ Namensraum-Deklaration (xml-namespace) (optional)
❻ Element (`Name="titel"`; mit Inhalt, hier: Zeichendaten)
Attribut aus ❼ Attributname und ❽ Attributwert
❾ Entity (`&entityname;`), hier: `amp = ampersand = "&"`
❿ Element (`Name="strophe"`; Inhalt: mixed content !)
⓫ Element (`Name="br"`; leeres Element)
⓬ Kommentar

Regeln

- Elemente müssen geschlossen werden:
  `<elementname>Inhalt</elementname>` (Öffnendes und schließendes Tag) oder `<elementname />` (Leeres Element)
- Alle Elemente müssen sauber geschachtelt sein (keine Überlappung! **Ein** Wurzelelement!)
- Elementnamen beachten Groß-/Kleinschreibung
- Attribute dürfen sich in einem Element nicht wiederholen; Attributwerte in Anführungszeichen
- Entities z.B. für "<" `(&lt;)`, ">" `(&gt;)`, "&" `(&amp;)` – man kann Entities auch selbst definieren
- Elemente können bestimmten Namensräumen angehören (werkzeug:mutter ≠ familie:mutter); Notation: *namensraum:elementname*

Terminologie

- Ein XML-Dokument, das die grundlegenden XML-Regeln erfüllt, ist **wohlgeformt** (well-formed).
- Ein XML-Dokument, das die Regeln eines Schemas erfüllt, ist **gültig** (valid).

DTD – Document Type Definition

DTD ist ein Beispiel für eine Schemasprache!
Weitere Schemasprachen sind z.B. XMLSchema (XSD) oder RelaxNG. Ein Schema beschreibt das Vokabular und die Regeln einer Auszeichnungssprache (ihre "Grammatik").

Beispiel

```
<!ELEMENT gedicht (titel?, strophe+)>
<!ELEMENT titel (#PCDATA)>
<!ATTLIST titel lang   (de | en | fr) REQUIRED
                n      (CDATA)        IMPLIED>
<!ELEMENT strophe (#PCDATA | br)*>
<!ELEMENT br EMPTY >
```

allgemein:

```
<!ELEMENT Elementname (Inhaltsmodell)>
<!ATTLIST Elementname Attributname
                (Attributwerte) Verbindlichkeit>
```

Regeln

- Abfolge: `,` bedeutet festgelegte Reihenfolge ("dann");
  `|` bedeutet beliebige Reihenfolge ("oder")
- Zeichendaten in Elementen: `#PCDATA`
- Quantoren (bei Elementen, Gruppen, Modellen):
  nichts = genau ein Mal
  `?` = kein oder ein Mal
  `+` = ein oder mehrere Mal
  `*` = kein, ein oder mehrere Mal
- Mixed Content: `(#PCDATA | elementname)*`
- Gruppen in Inhaltsmodellen durch "(...)"
- Attributwerte nicht festgelegt: `CDATA`
- Verbindlichkeit von Attributen: `IMPLIED` = "kann vorkommen"; `REQUIRED` = "muss vorkommen"; `FIXED` = "fester Wert"
- Kommentar: `<!-- Text -->`

Wichtige XPath-Funktionen

`name()` bzw. `local-name()` – Name des aktuellen Knotens

`count(XPath)` – Anzahl der Knoten in einer Knotenmenge

`position()` – Position eines Knotens in einem Set, abgekürzte Syntax als Prädikat: [*n*]

`string(object?)` bzw. `number(object?)`
– konvertiert etwas explizit in einen String oder eine Zahl

`string-length(string?)` – Länge der Zeichenkette

`concat(string,string,string*)` – Verknüpfung mehrerer Strings

`contains(string,string)` – enthält der erste den zweiten String?

`starts-with(string,string)`
– beginnt der erste String mit dem zweiten?

XPath – XML Path Language

XPath ist eine Sprache zur Navigation in XML-Dokumenten und zur Adressierung und Selektion von "Knoten".
XPath-Ausdrücke können zusammengesetzt sein aus Pfadangaben (Lokalisierungsschritte, Knotentests), Bedingungen (Prädikaten, "[...]") und Funktionen ("*function*()").
Lokalisierungsschritte folgen vertikalen Achsen (Baumstruktur / Hierarchie, Attribute) oder horizontalen Achsen (Reihenfolge im Dokument).

Beispiele

`//*` – beliebige Elemente in beliebiger Tiefe

`/gedicht/strophe/br`
– Pfad zu Element `br` als Kind von `strophe` als Kind von `gedicht`

`count(//gedicht/*)`
– Anzahl aller Kinder von allen Elementen `gedicht`

`/gedicht/titel[@lang='de'][1]`
– von allen `titel`, die Kind von `gedicht` sind und deren Attribut `lang` den Wert "de" haben, das erste (`position()='1'`)

`//titel[contains(.,'Kreti')]`
– alle Elemente `titel`, die die Zeichenkette "`Kreti`" enthalten

`//index[not(preceding::index=.)]`
– alle Elemente `index`, zu denen es kein vorhergehendes Element `index` mit dem gleichen Inhalt gibt (→ nur das erste von gleichen)

Kurz	Achse	Bedeutung
.	self	der aktuelle Kontextknoten
nichts	child	direkt untergeordnete Knoten
..	parent	direkt übergeordneter Knoten
	ancestor	übergeordnete Knoten
	ancestor-or-self	dito, plus Kontextknoten
	descendant	untergeordnete Knoten
//	descendant-or-self	dito, plus Kontextknoten
	following	nachfolgend im Dokument
	following-sibling	nachfolgend im gleichen parent
	preceding	vorhergehend im Dokument
	preceding-sibling	vorhergehend im gleichen parent
@	attribute	Attributknoten

`substring(string,number,number?)`
– Extrahiert aus string ab einer Position (erste number) eine bestimmte Menge Zeichen (zweite number)

`substring-before(string,string)` bzw.
`substring-after(string,string)`
– Teil des ersten Strings vor/nach dem Vorkommen des zweiten Strings

`translate(string,string,string)`
– ersetzt im ersten String alle Zeichen des zweiten Strings durch das Zeichen an der gleichen Stelle im dritten

`matches(string,regex,flags*)`
– untersucht einen String mit regulären Ausdrücken (XPath2.0!)

Gestaltung: Katharina Weber