tekom Jahrestagung 11
mit tcworld conference tcworld conference 2011

<Tutorial>

# XSLT-Programmierung – effektiv und schmerzfrei!

Dr. Thomas Meinike
Hochschule Merseburg

thomas.meinike@hs-merseburg.de
http://www.iks.hs-merseburg.de/~meinike/
@XMLArbyter

19. Oktober 2011 / Wiesbaden

# 0. Motivation und Einstieg

**@mrmixin**
Mike Turner ツ

XSLT is one of those things that doesn't work when you think you understand it, but does work when you don't know what's going on. #XSLT

# 0. Motivation und Einstieg

## XSLT …

… steht für Extensible Stylesheet Language Transformations

… ist eine seit > 10 Jahren etablierte Technologie zur XML-Verarbeitung

… wird vom W3C spezifiziert

… ist angeblich schwierig zu erlernen und zu nutzen (jein ;)

… polarisiert zumindest die Gemüter, siehe Amplicate.com:


[16. Oktober 2011]

… soll in diesem Tutorial aus anwendungsorientierter Sicht behandelt und mit praktischen Beispielen demonstriert werden

@braegel
braegel

Thinking about learning XSLT.

# 1. Grundausstattung

**@m1ke_wallace**
Michael Wallace

Who remembers XSLT?

# 1. Grundausstattung

## Allgemeines

- Transformationssprache für XML-Dokumente in Zielformate auf der Basis von HTML, XML und Text
- XML-basierte Sprache, arbeitet mit Templates (= funktionale Einheiten), deklarativ-funktionale Sprache
- Gehört mit XSL-FO und XPath zur XSL-Technologiefamilie
- XPath wird zur Lokalisierung von Knoten und Inhalten innerhalb von XML-Strukturen verwendet
- XSL-FO ermöglicht Druckausgaben (PDF), wird oft mit XSLT kombiniert
- W3C-Hauptspezifikationen zu XSLT / XPath: 1.0 (1999) sowie 2.0 (2007) Erweiterungen 2.1 / 3.0 in Arbeit

# 1. Grundausstattung

## XSLT-Grundgerüst (Version 2.0)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:fn="http://www.w3.org/2005/xpath-functions"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  exclude-result-prefixes="fn xs">

  <!-- Ausgabeoptionen -->
  <xsl:output method="…" version="…" encoding="…" indent="…"
              doctype-public="…" doctype-system="…"/>

  <!-- Haupt-Template (für Wurzelelement oder Wurzelknoten "/") -->
  <xsl:template match="element_1">
    <!-- Aufruf eines weiteren Templates -->
    <xsl:apply-templates select="element_2"/> <-------- <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="element_2">
    <!-- weiterer Code -->
  </xsl:template>

</xsl:stylesheet>
```

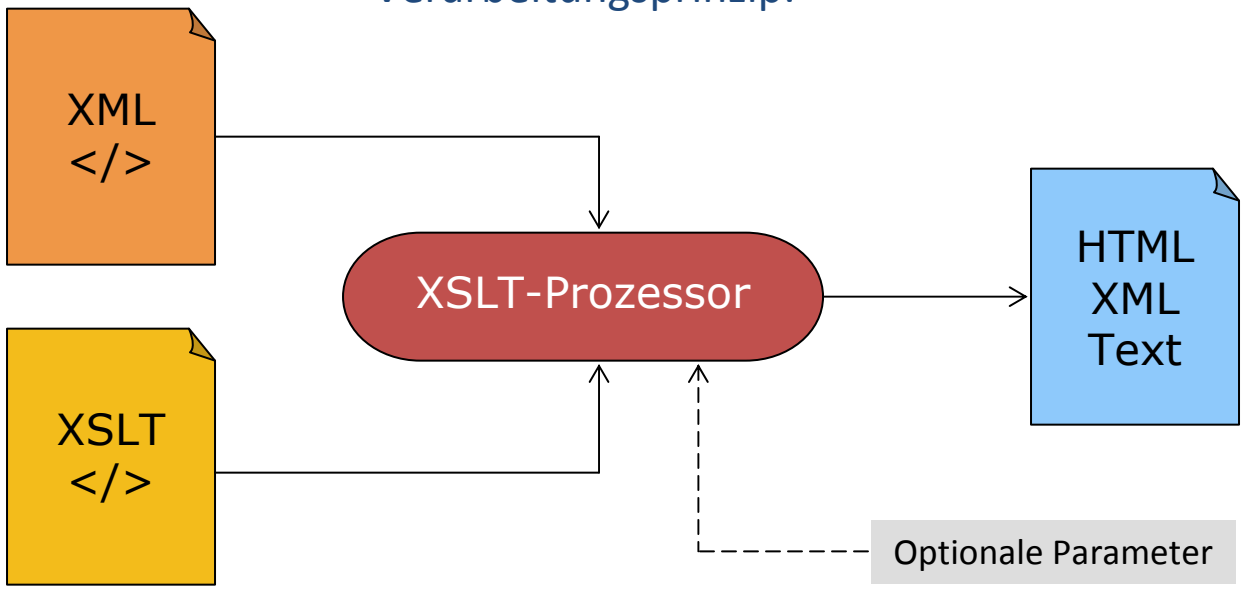
method:
html
text
xml
xhtml (2.0)

## Transformationen

- XML-Dokumente werden mit XSLT-Stylesheets unter Verwendung von XSLT-Prozessoren in Zielformate überführt

- XSLT-Prozessoren sind eigenständige Programme (.exe, .jar) oder Bibliotheken (.dll) und werden in Redaktionssysteme sowie XML-Werkzeuge integriert

Verarbeitungsprinzip:



# 1. Grundausstattung

## XSLT-2.0-Prozessoren und Konsolen-Aufrufsyntax

- Saxon 9.3 (HE / PE / EE) von Michael Kay
  http://saxon.sourceforge.net/
  **java -jar saxon9he.jar -s:name.xml -xsl:name.xsl -o:name.out**

- AltovaXML 2011 Community Edition
  http://www.altova.com/altovaxml.html
  **AltovaXML.exe -in name.xml -xslt2 name.xsl -out name.out**

- Intel SOA Expressway XSLT 2.0 Processor Beta 2
  http://software.intel.com/en-us/articles/intel-soa-expressway-xslt-20-processor/
  **soaexslt2.exe -o name.out name.xsl name.xml**

- XQSharp 2.2 (XML Processing for the .NET Framework)
  http://www.xqsharp.com/
  **XSLT.exe -o name.out -s name.xml name.xsl**

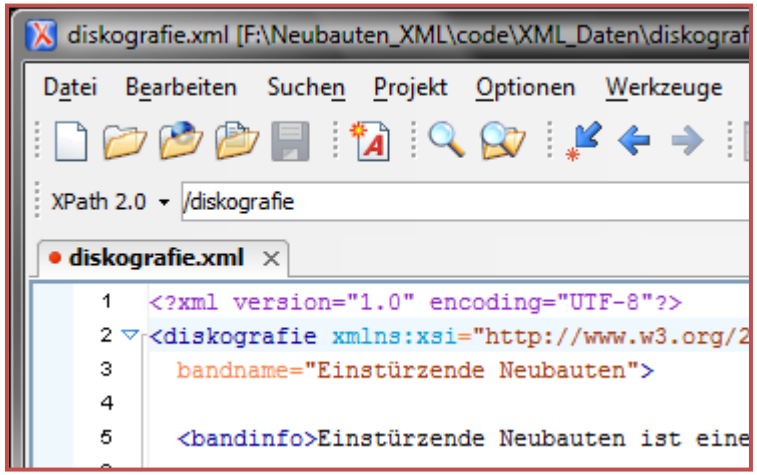
**@mpuckett**
Michael Puckett

endlessly frustrated that .net doesn't natively support xslt 2.0

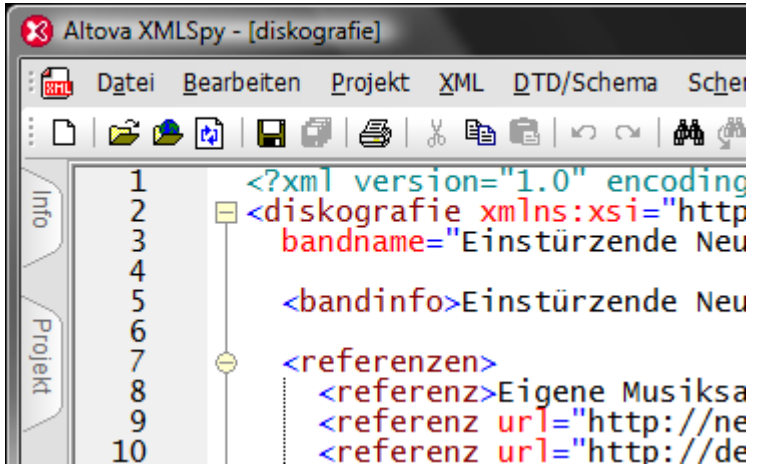
# 1. Grundausstattung

## XML-Editoren mit XSLT-Unterstützung (Auswahl)

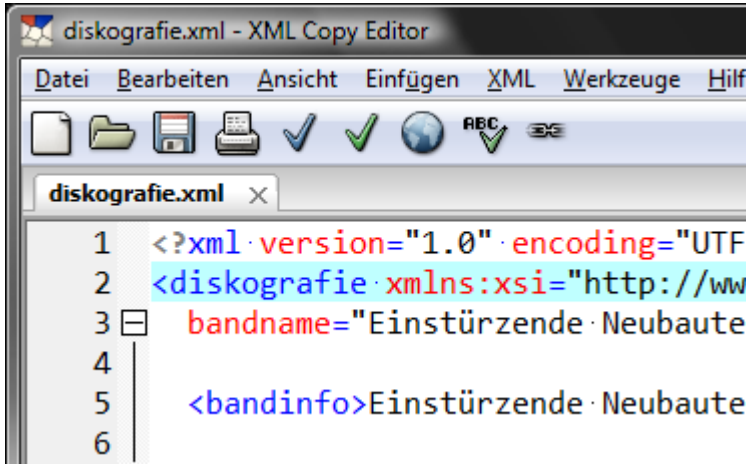
<oXygen/> (komm.)



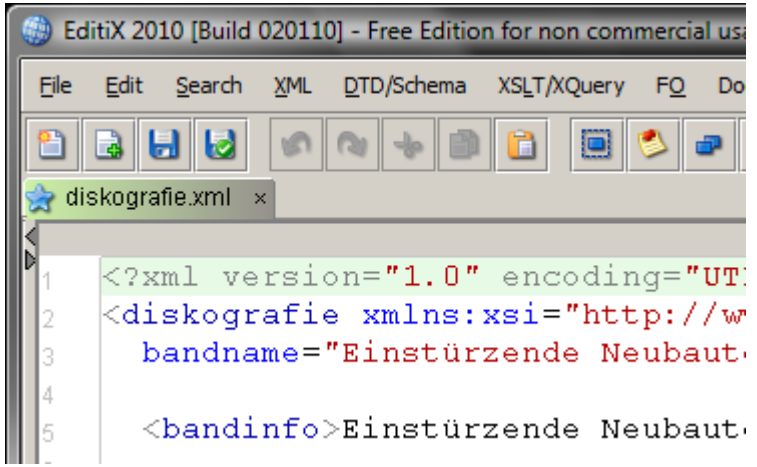

XMLSpy (komm.)




XML Copy Editor (frei)

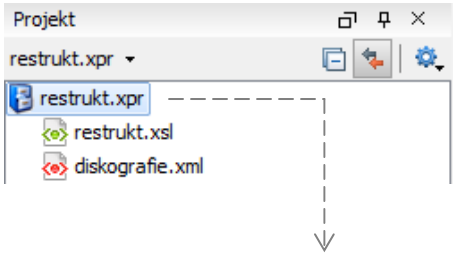



EditiX (frei)



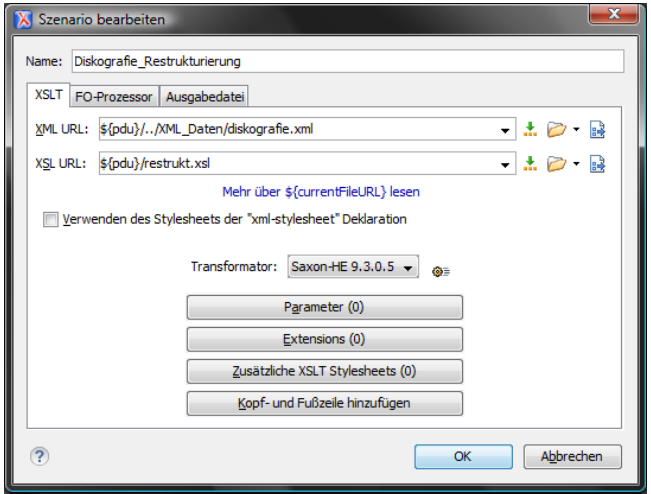
## Nutzung von XSLT / FO im <oXygen/> XML Editor

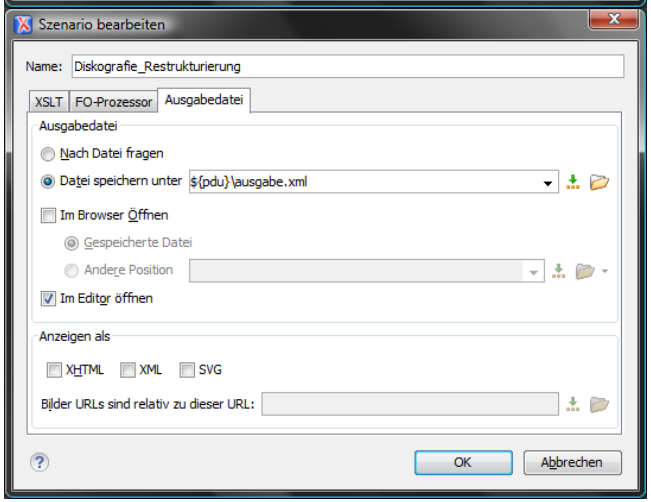
- Projekt (.xpr) anlegen mit Referenzen zu XML, XSLT, DTD, Schema, …




- Transformationsszenario konfigurieren:
  - ✓ Prozessor wählen (z. B. Saxon 9.3)
  - ✓ ggf. zusätzlich FO-Formatierer wählen
  - ✓ Ein- und Ausgaben festlegen
  - ✓ Anzeige einstellen (z. B. im Browser)
  - ✓ Nutzung von Variablen für Namen und Pfade möglich

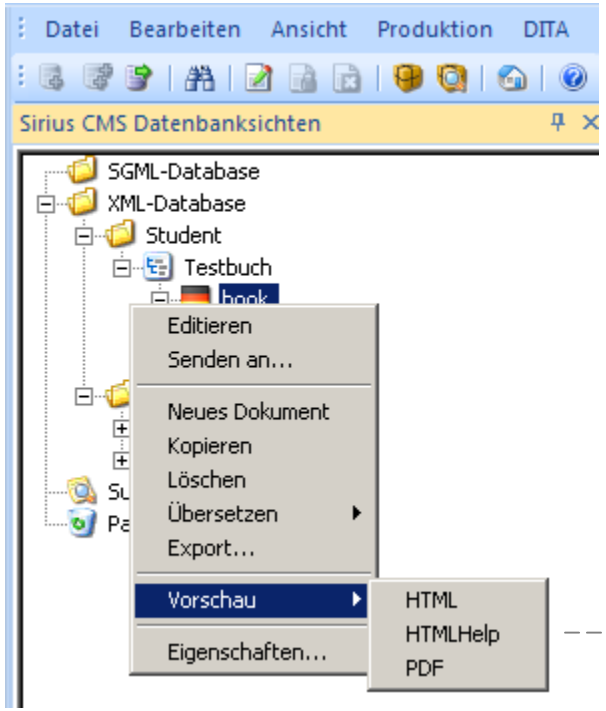
- Transformation auf Knopfdruck  oder über Kontextmenü starten



# 1. Grundausstattung

## XSLT / FO als »Black Box« in ein Redaktionssystem integriert




Acolada Sirius CMS

Verweise auf im System hinterlegte XSLT-Stylesheets und Verarbeitung zu den jeweiligen Zielformaten auf »Knopfdruck« …

# 1. Grundausstattung

## Sonstige XSLT-Nutzung in Anwendungen

- Adobe FrameMaker
- Adobe InDesign
- Clientseitige Transformationen im Web-Browser
- Office-Software (Microsoft Office, OpenOffice)
- Serverseitige Transformationen (PHP & Co.)
- Toolkits (DITA OT, DocBook-Stylesheets)
- …

**@eddieforeman**
Eddie Foreman

Xslt favourite movie = lost in transformation

# 1. Grundausstattung

## Das Beispielprojekt – Übersicht

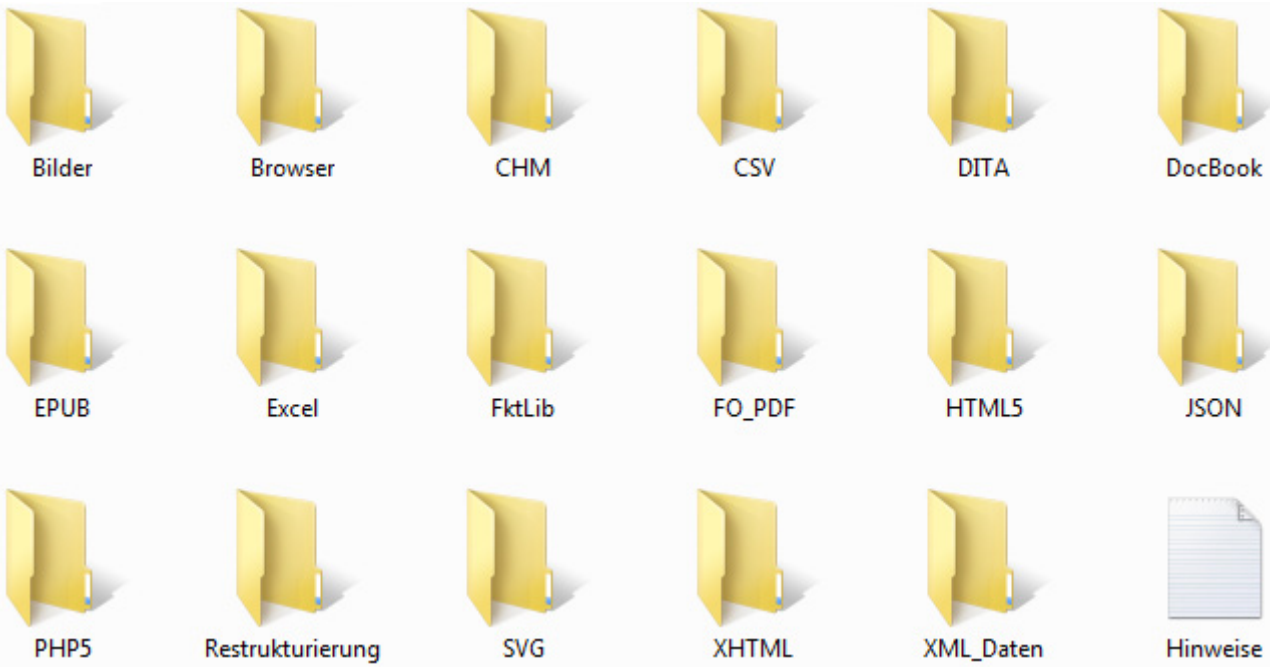
- Beschreibung der Diskografie einer beliebigen Band mit XML-Schema sowie DTD als Datenmodelle (diskografie.xsd / diskografie.dtd)
- XML-Instanz diskografie.xml mit 30 Datensätzen (= Werke)
- Transformationen in unterschiedliche Zielformate (Single-Source-Publishing):
  - ✓ Browser
  - ✓ CHM
  - ✓ CSV
  - ✓ DITA
  - ✓ DocBook
  - ✓ EPUB
  - ✓ Excel
  - ✓ FO/PDF
  - ✓ HTML5
  - ✓ JSON
  - ✓ Restrukturierung
  - ✓ PHP5
  - ✓ SVG
  - ✓ XHTML
- Ziel: Vorstellung der verwendeten Techniken

@matthewkimber
matthewkimber

Swimming in a sea of XML and XSLT.

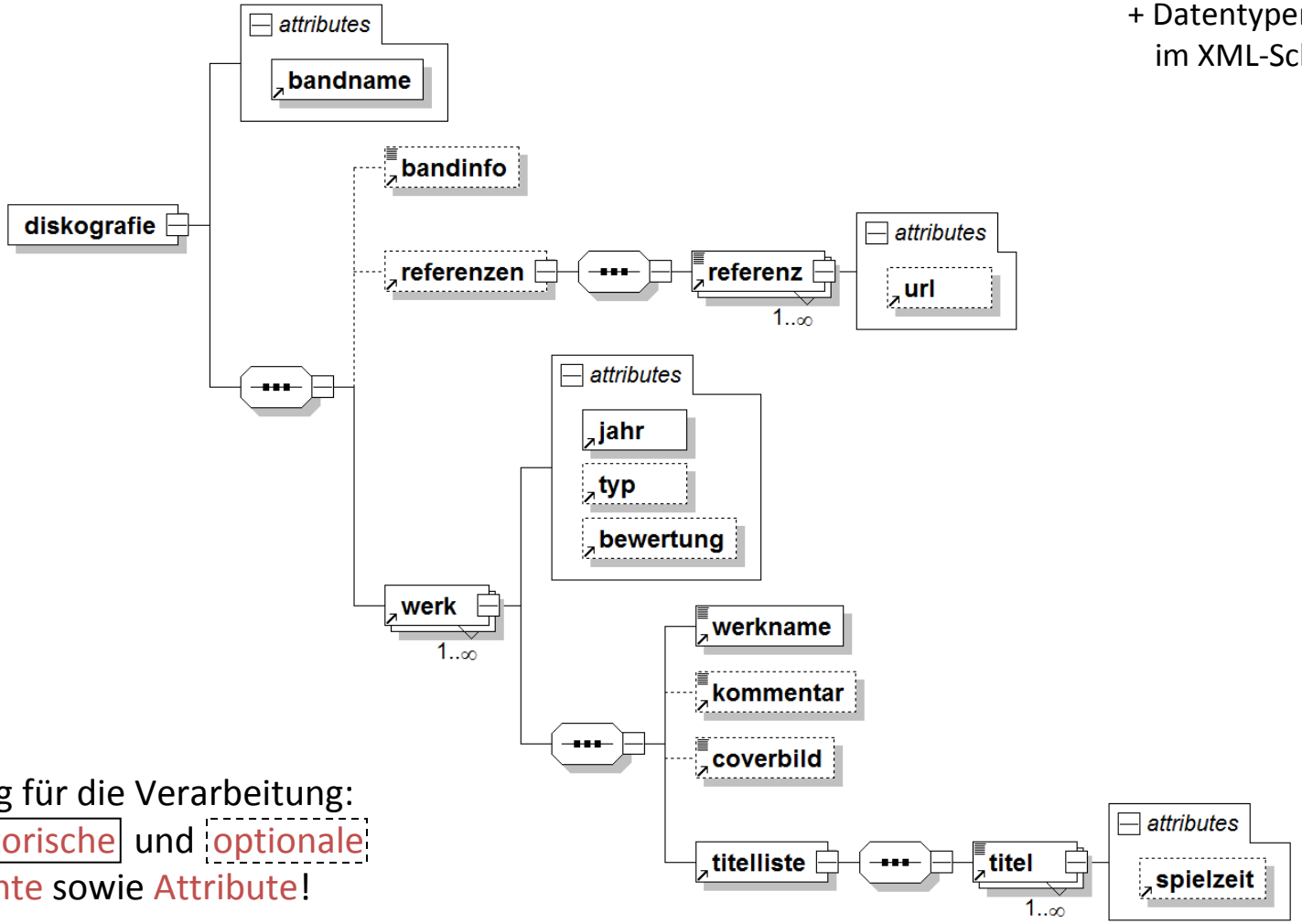
# 1. Grundausstattung

## Das Beispielprojekt – Verzeichnisstruktur



## Das Beispielprojekt – Dokumentstruktur

+ Datentypen
  im XML-Schema




Wichtig für die Verarbeitung:
obligatorische und optionale
Elemente sowie Attribute!

## Das Beispielprojekt – Instanzdokument

```
<?xml version="1.0" encoding="UTF-8"?>
<diskografie xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="diskografie.xsd" bandname="Einstürzende Neubauten">

  <bandinfo>Einstürzende Neubauten ist eine deutsche experimentelle Band. ...</bandinfo>

  <referenzen>
    <referenz>...</referenz><referenz url="...">...</referenz>
  </referenzen>

  <werk jahr="1987" typ="LP" bewertung="9">
    <werkname>Fünf auf der nach oben offenen Richterskala</werkname>
    <kommentar>Der Song Morning Dew ist eine Coverversion ...</kommentar>
    <coverbild>en_richterskala.jpg</coverbild>
    <titelliste>
      <titel spielzeit="8:01">Zerstörte Zelle</titel>
      <titel spielzeit="4:57">Morning Dew</titel>
      <titel spielzeit="3:22">Ich bin's</titel>
      <titel spielzeit="4:50">Mo Di Mi Do Fr Sa So</titel>
      <titel spielzeit="7:32">12 Städte</titel>
      <titel spielzeit="3:06">Keine Schönheit ohne Gefahr</titel>
      <titel spielzeit="6:42">Kein Bestandteil sein</titel>
      <titel spielzeit="5:42">Bonus: Adler kommt später (remastered CD 2002)</titel>
    </titelliste>
  </werk>

</diskografie>
```

Hauptteil = 30 werk-Elemente

+ 30 JPEG-Coverbilder

# 2. Der funktionale Ansatz

**@SeidoKevin**
Kevin Bronsdijk

Working on some XSLT templates and having a hard time moving from a procedural mindset to a declarative.

# 2. Der funktionale Ansatz (1)

## Grundsätzliches

- Templates (xsl:template) sind funktionale Einheiten, sie werden separat verarbeitet und ihre Ergebnisse ohne Seiteneffekte in den Ergebnisbaum geschrieben (serialisiert)
- Insofern findet zwischen Templates keine direkte Kommunikation außer ihrem mit XPath-gesteuerten Aufruf (xsl:apply-templates / xsl:call-template) statt
- Mit xsl:call-template lassen sich jedoch pro Aufruf Parameter zwischen Templates übergeben (ermöglicht rekursive Verarbeitung von Algorithmen)
- Variablen und Parameter »(über)leben« nur in ihrer jeweiligen Template-umgebung (Scope) – lokal oder global unterhalb von xsl:stylesheet
- In prozeduralen Sprachen übliche Operationen wie $x = $x + 1 bzw. $x++ lassen sich also nur über Umwege erreichen (neuere 2.0-Techniken vereinfachen jedoch einfache Abzählvorgänge, siehe 1 to $n)

# 2. Der funktionale Ansatz (2)

## Spezielles

- Push-Verfahren:
  - ✓ Datengetriebene Ablaufkontrolle liegt beim Prozessor
  - ✓ Nutzung von xsl:template und xsl:apply-templates (ohne bzw. mit select-Attribut)
  - ✓ Verarbeitung wird an das jeweils »zuständige« Template delegiert
  - ✓ Ermöglicht flexiblere Wiederverwendung von Templates
  - ✓ Modi für erweiterte Template-Steuerung (mode-Attribut)

- Pull-Verfahren:
  - ✓ Ablaufkontrolle liegt im Code (Anordnung der Anweisungen)
  - ✓ Direkte Abarbeitung von Knoten mit xsl:for-each / xsl:value-of und Abfragen
  - ✓ Bietet sich oft zur sequentiellen Ausgabe von listenartigen Inhalten an
  - ✓ Kann die Wiederverwendung von Templates erschweren, also mit Bedacht einsetzen (die folgenden Beispiele sind bewusst nicht völlig frei davon …)

- Gängige Praxis:
  - ✓ In Abhängigkeit von der konkreten XML-Dokumentstruktur die geeignete Strategie wählen und nach Möglichkeit separate Templates bevorzugen

# 3. XPath im Überblick

**@derSachse**
Stefan

Sehr genial. XPath ist ganz schön kraftvoll.

# 3. XPath im Überblick

## Lokalisierung und Zugriff auf dem XML-Baum

- Lokalisierungspfade analog zu Pfaden bei Dateisystemen mit einzelnen Lokalisierungsschritten /…/ sowie zusätzlichen Prädikaten […]:

| | |
|---|---|
| Elementzugriff: | /diskografie/werk/werkname<br>/diskografie/werk[10]/werkname |
| Attributzugriff: | /diskografie/werk/@typ |
| Prädikatabfrage: | /diskografie/werk[@jahr = 1981]/werkname<br>/diskografie/werk[@typ = 'LP']/@jahr |

- Praktikabel wird der Zugriff auf die einzelnen Hierarchieebenen über den Template-Mechanismus: Template für *diskografie* ruft Template für *werk* auf und dieses verarbeitet die einzelnen *werk*-Strukturen und -Inhalte (ruft ggf. weitere Templates oder andere Konstrukte auf)

- Zusätzlich kann die Verarbeitungsrichtung mittels Achsen gesteuert werden: child:: (Standardachse ohne Angabe), parent:: (kurz **..**), attribute:: (kurz **@**), self:: (Kontextknoten, kurz **.**), ancestor::, descendant::, following::, preceding::, …

# 3. XPath im Überblick

## XPath-Funktionen

- Erweitern die Möglichkeiten zur Datenabfrage und -verarbeitung (kleine Auswahl aus > 100):

| | |
|---|---|
| fn:string-length($text) | fragt die Zeichenzahl von $text ab |
| fn:concat ($a, $b, $c) | fügt Zeichenketten zusammen |
| fn:substring($text, 1, 10) | gibt 10 Zeichen ab Pos. 1 zurück |
| fn:substring-before($text, 'abc') | holt Text vor 'abc' |
| fn:substring-after($text, 'abc') | holt Text nach 'abc' |
| fn:number($text) | numerischen Wert aus Zeichenkette bilden |
| fn:count(…) | Anzahl innerhalb einer Sequenz im akt. Kontext |
| fn:position() | akt. Position im Template oder in xsl:for-each |
| fn:current-date() | akt. Systemdatum auslesen |
| fn:replace(…) | Ersetzungen mittels regulärer Ausdrücke |

- XPath-Ausdrücke und Funktionen werden innerhalb der Attribute match, select (Abfragen) und test (Prüfungen) verwendet

# 4. Kontrollstrukturen, Datentypen, Operatoren, Variablen und Parameter

**@peterhefley**
peterhefley

Steps to Success [with XSLT] : 1) Drink beers. 2) Gain perspective. 3) Add missing dollar sign. 4) Profit.

# Kontrollstrukturen (1)

- Einfache Prüfungen mit xsl:if und test-Attribut
- Im FO-Beispiel:
  - ✓ Ausgabe der Referenzen, wenn es das optionale Element referenzen gibt

```
<xsl:if test="referenzen">
  <fo:block ...>
    <xsl:text>Referenzen</xsl:text>
  </fo:block>
  <xsl:apply-templates select="referenzen/referenz"/>
</xsl:if>
```

- In den HTML-Beispielen:
  - ✓ h3-Ausgabe: 1981 (LP), sofern ein typ-Attribut mit Inhalt exisitert, sonst nur 1981 (@jahr → Pflichtattribut, @typ → optional, könnte leer sein, deshalb Frage nach > 0)

```
<h3>
  <xsl:value-of select="@jahr"/>

  <xsl:if test="fn:string-length(@typ) > 0">
    <xsl:value-of select="fn:concat(' (', @typ, ') ')"/>
  </xsl:if>
</h3>
```

# Kontrollstrukturen (2)

- Mehrfache Prüfungen mit xsl:choose / xsl:when / xsl:otherwise und test-Attribut bei xsl:when

- In den HTML-Beispielen:
  - ✓ Ausgabe der Referenzen als Listeneinträge (li-Element) und sofern URL vorhanden verlinkter Text (a-Element), ansonsten nur Text (@url → optionales Attribut)

```
<xsl:template match="referenz">
  <li>
    <xsl:choose>
      <xsl:when test="fn:string-length(@url) > 0">
        <a href="{@url}"><xsl:value-of select="."/></a>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="."/>
      </xsl:otherwise>
    </xsl:choose>
  </li>
</xsl:template>
```

- Hinweis: XPath 2.0 erlaubt inzeilige if-then-else-Konstrukte

## Datentypen

- Standardtypen sind string für Textinhalte und number für Zahlenwerte
- XPath 2.0 erlaubt genauere Zuweisungen wie xs:integer, xs:double, xs:date, …

## Operatoren

- Rechenoperatoren: + – * div idiv mod
- Vergleichsoperatoren: < <= >= > != mit < für < (ab 2.0 auch kurz lt)
- Logikoperatoren: and or not()

## Variablen / Parameter

- Sind als Konstanten aufzufassen (beliebig, aber nach Zuweisung fest)
- xsl:variable bzw. xsl:param mit Elementinhalt oder select-Attribut
- Parameter lassen sich zwischen Templates bewegen und auch von außen einfügen

# Templates rekursiv aufrufen (1)

- Steuerung mit xsl:template / xsl:call-template / xsl:with-param / xsl:param
  - ✓ FO-Ausgabe von Sternen entsprechend der Bewertung (1–10):

```
<xsl:if test="@bewertung >= 1">
  <fo:inline alignment-adjust="1pt">
    <xsl:call-template name="sterne">
      <xsl:with-param name="wert" select="@bewertung"/>
    </xsl:call-template>
  </fo:inline>
</xsl:if>


<xsl:template name="sterne">
  <xsl:param name="wert"/>

  <xsl:if test="$wert > 0">
    <fo:external-graphic src="url('{$bildpfad}/stern.gif')" content-height="10pt"/>
    <xsl:call-template name="sterne">
      <xsl:with-param name="wert" select="$wert - 1"/>
    </xsl:call-template>
  </xsl:if>
</xsl:template>
```

Einfacher lösbar?!

# Templates rekursiv aufrufen (2)

- Alternative ab 2.0: Zählsequenzen wie (1 to $n) mit xsl:for-each

```
<xsl:if test="@bewertung >= 1">
  <fo:inline alignment-adjust="1pt">

    <xsl:for-each select="1 to @bewertung">
      <fo:external-graphic src="url('{$bildpfad}/stern.gif')" content-height="10pt"/>
    </xsl:for-each>

  </fo:inline>
</xsl:if>
```

# 5. Elemente und Attribute erzeugen

**@natpryce**
Nat Pryce

Does anyone *not* find XSLT to be a write only language?

# 5. Elemente und Attribute erzeugen

## Elemente

- Ohne Abhängigkeit von sich dynamisch ergebenden Bezeichnern oder weiteren Operationen grundsätzlich literal erzeugen, also direkt in den Ausgabebaum schreiben (kompakter, gut les- und wartbarer Code)!

```
<p>Kommentar: <em><xsl:value-of select="kommentar"/></em></p>
```

- Variante mit xsl:element bringt keine Vorteile, ist schlecht les- und wartbar!

```
<xsl:element name="p">
  <xsl:text>Kommentar: </xsl:text>
  <xsl:element name="em">
    <xsl:value-of select="kommentar"/>
  </xsl:element>
</xsl:element>
```

- Sinnvoller Einsatz von xsl:element (variabler Elementname):

```
<xsl:element name="{XPath-Ausdruck}">
  <!-- Elementinhalt -->
</xsl:element>
```

→ siehe Beispiel Restrukturierung

# 5. Elemente und Attribute erzeugen

## Attribute

- Ebenfalls möglichst literal notieren, {…} = Attribute Value Template (AVT)

```
<img src="{$bildpfad}/{coverbild}" alt="Coverbild" />
```

- Variante mit xsl:attribute ist wiederum wenig sinnvoll (man könnte hier auch noch img mittels xsl:element konstruieren …)

```
<img>
  <xsl:attribute name="src">
    <xsl:value-of select="fn:concat($bildpfad, '/', coverbild)"/>
  </xsl:attribute>
  <xsl:attribute name="alt">
    <xsl:text>Coverbild</xsl:text>
  </xsl:attribute>
</img>
```

- Sinnvoller Einsatz von xsl:attribute (dynamischer Bezeichner und / oder Inhalt):

```
<xsl:attribute name="{XPath-Ausdruck}">
  <!-- Attributinhalt, ggf. bei festem Namen auch dynamisch erzeugt -->
</xsl:attribute>
```

→ siehe Beispiel Restrukturierung

# 6. Sortieren, Nummerieren, Gruppieren

**@euperia**
Andrew McCombe

XSLT: Making things 3000 times harder than they need to be.

# 6. Sortieren, Nummerieren, Gruppieren

## Sortieren

- Element xsl:sort
  - ✓ Unterhalb von xsl:apply-templates, xsl:for-each, xsl:for-each-group platzierbar
  - ✓ select → Sortierkriterium (z. B. Inhalt von Element / Attribut)
  - ✓ data-type → text (Vorgabe) oder number
  - ✓ order → ascending (Vorgabe) oder descending
  - ✓ Typische Anwendung in den Beispielen:

```
<xsl:apply-templates select="werk">
  <xsl:sort select="@jahr" order="ascending" data-type="number"/>
</xsl:apply-templates>
```

## Nummerieren

- Element xsl:number
  - ✓ Gibt laufende Nummer in “document order” aus
  - ✓ format → 1 (Vorgabe) 01 a A I i (weitere Attribute wie count, level, …)

```
<xsl:number format="1"/>
```

  - ✓ Alternativen: fn:position() oder direkt im Markup, z. B. für HTML ol + li

# 6. Sortieren, Nummerieren, Gruppieren

## Gruppieren

- Element xsl:for-each-group
  - ✓ Neu in XSLT 2.0 (zur Muench-Methode mit xsl:key siehe svg_gm.xsl)
  - ✓ select → Was soll gruppiert werden?
  - ✓ group-by → Wonach soll gruppiert werden?
  - ✓ current-group() → Zugriff auf die einzelnen Gruppen
  - ✓ Einsatz im SVG-Beispiel:

```
<xsl:for-each-group select="werk" group-by="@jahr">
  <xsl:sort select="@jahr" order="ascending" data-type="number"/>
  <!-- ...  -->
  <!-- Werk-Titel (optional Typ), ggf. mehrere getrennt und ××× als Trennung -->
  <text x="90" y="{$akt_y + 5}">
    <xsl:for-each select="current-group()">
      <xsl:value-of select="werkname"/>
      <xsl:if test="fn:string-length(@typ) > 0">
        <xsl:value-of select="fn:concat(' (', @typ, ')')"/>
      </xsl:if>
      <xsl:if test="fn:position() != last()">
        <tspan fill="#00C" font-weight="bold"> ××× </tspan>
      </xsl:if>
    </xsl:for-each>
  </text>
</xsl:for-each-group>
```

# 7. Umgang mit Namensräumen bei Ein- bzw. Ausgaben

**@singlesoliloquy**
Tim Lepczyk

XSLT code was doing the wrong thing.

# 7. Umgang mit Namensräumen bei Ein- bzw. Ausgaben

## Angabe von Namensräumen

- In den XML-Quellen vorhandene xmlns:xyz="…" in das XSLT-Stylesheet übernehmen und über die jeweiligen Präfixe ansprechen
- Der Vorgabe-Namensraum in der Eingabe (xmlns="…") kann unter XSLT 2.0 bei xsl:output als xpath-default-namespace deklariert werden
- Namensräume für die Ausgabe ebenfalls im Stylesheet angeben
- Unerwünschte Ausgabe-Präfixe mit exclude-result-prefixes steuern
- Im Excel-Beispiel (SpreadsheetML):

```
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:fn="http://www.w3.org/2005/xpath-functions"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns="urn:schemas-microsoft-com:office:spreadsheet"
  xmlns:ss="urn:schemas-microsoft-com:office:spreadsheet"
  xmlns:x="urn:schemas-microsoft-com:office:excel"
  exclude-result-prefixes="fn xs">

  <!-- ... -->

</xsl:stylesheet>
```

# 8. Von XSLT / XPath 2.0 profitieren

**@XMLArbyter**
ThomasM

xsl:result-document ist mein Mitarbeiter des Monats. #XSLT

# 8. Von XSLT / XPath 2.0 profitieren

## Mehrfachausgaben

- Element xsl:result-document
- Steuerung über Attribute href und format (Zuordnung mit xsl:output / name)

```
<xsl:result-document href="..." format="...">
  <!-- ... -->
</xsl:result-document>
```

- In den Beispielen für CHM, DITA und EPUB verwendet

## Kompakte Formulierungen

- Inline-XPath-Konstrukte (flexibler, sparen Code, z. B. rekursive Aufrufe):
  - ✓ if … then … else
  - ✓ for … in … return
  - ✓ to-Operator zum Abzählen: 1 to $n
- In den Beispielen u. a. für Zeitberechnungen und Ausgabe der Sterne eingesetzt

# 8. Von XSLT / XPath 2.0 profitieren

## Eigene, wieder verwendbare, Funktionen

- Element xsl:function
  - ✓ Benötigen eigenen Namensraum (mit Präfix, hier tm)
  - ✓ Datentypen für Ein- und Ausgabe werden mit as-Attribut definiert
  - ✓ Argumente werden über xsl:param-Elemente abgebildet
  - ✓ Anwendung der eigenen Funktionen analog zu den vordefinierten
  - ✓ Anlage von Funktionsbibliotheken und Einbindung mit xsl:include / xsl:import
  - ✓ Einsatz zur Zeitberechnung (Summe der Spielzeiten) in den Beispielen:

```
<!-- Ermittelt aus einer Sequenz ('1:11','2:23','3:42') die Ausgabe (00:07:16) -->
<xsl:function name="tm:gesamtzeit" as="xs:string"><!-- alternativ as="xs:time" -->

  <xsl:param name="zeitstr" as="xs:string*"/>

  <xsl:variable name="s" select="fn:sum(for $t in $zeitstr return
    60 * fn:number(fn:substring-before($t, ':')) + fn:number(fn:substring-after($t, ':')))"/>

  <xsl:value-of select="xs:time('00:00:00') + $s * xs:dayTimeDuration('PT1S')"/>

</xsl:function>


<!-- Aufruf: -->
<xsl:value-of select="tm:gesamtzeit(...)"/>
```

FktLib/library.xsl

# 8. Von XSLT / XPath 2.0 profitieren

## Datums- und Zeitfunktionen

- Systemdatum und -zeit auslesen (XPath-Funktionen)
  - ✓ fn:current-date()
  - ✓ fn:current-time ()
  - ✓ fn:current-dateTime()
- Formatierungen (XSLT-Funktionen)
  - ✓ format-date()
  - ✓ format-time()
  - ✓ format-dateTime() und Formatstring
- Im Excel-Beispiel verwendet mit UTC-Anpassung:
  - ✓ fn:adjust-dateTime-to-timezone()
  - ✓ xs:dayTimeDuration() und Zeitperiode (PT0H)

```
<Created>
  <xsl:value-of select="format-dateTime(fn:adjust-dateTime-to-timezone(fn:current-
    dateTime(), xs:dayTimeDuration('PT0H')), '[Y0001]-[M01]-D01]T[H01]:[m01]:[s01]Z')"/>
</Created>
```

↓

```
<Created>2011-09-26T16:28:17Z</Created>
```

# 9. Typische Anwendungsfälle

**@anlumo1**
Andreas Monitzer

Coding XSLT used to be fun for me, but now it's rather dull work...

# 9. Typische Anwendungsfälle

## Einsatz von XSLT im Dokumentationsbereich

- Umwandeln von XML-Dokumenten in andere XML-Formate:
  - ✓ XHTML (Web, Online-Hilfen, E-Books)
  - ✓ XSL-FO → PDF
  - ✓ EPUB (E-Books)
  - ✓ IDML (Adobe InDesign)
  - ✓ SVG (Vektorgrafiken)
  - ✓ SpreadsheetML (MS Excel), WordML (MS Word)
  - ✓ DITA, DocBook (spezielle Dokumentationsformate)
  - ✓ Newsfeeds (Atom, RSS)
  - ✓ Firmenspezifische Formate / Austauschformate / Kataloge …
- Umwandeln von XML-Dokumenten in Text-basierte Formate:
  - ✓ CSV (Comma Separated Values) für Datenbank-Anwendungen
  - ✓ JSON (JavaScript Object Notation) für Web-Anwendungen
  - ✓ Projektdatei (.hhp) für HTML Help (CHM)
  - ✓ …

@davecheney
Dave Cheney

I just used XSLT to solve a real world problem!

# 10. Demonstrationen und Diskussion

**@paolobozzelli**
Paolo Bozzelli

Improving XSLT Transformation is the new Sunday lunch.

# 10. Demonstrationen und Diskussion

## (1) XSLT im Browser

- Aktuelle Browser unterstützen nur XSLT / XPath 1.0
- Zuweisung von browser.xsl im XML-Dokument über Verarbeitungsanweisung (Processing Instruction) <?xml-stylesheet … ?>

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE diskografie SYSTEM "diskografie.dtd">
<?xml-stylesheet href="browser.xsl" type="text/xsl"?>
<diskografie bandname="Einstürzende Neubauten">
  <!-- ... -->
</diskografie>
```

- Hinweis: application/xslt+xml funktioniert noch nicht → text/xsl
- Demo im Verzeichnis Browser

# 10. Demonstrationen und Diskussion

## (2) XSLT auf dem Web-Server mit PHP

- PHP unterstützt XSLT 1.0 (Code ab 5.0, links transform.php, unten index.php)

```php
<?php
  function XSLTrans()
  {
    // XML-Dokument:
    $xmldocument = file_get_contents("diskografie.xml");

    // XSLT-Stylesheet:
    $xsldocument = file_get_contents("server.xsl");

    // Transformation:
    if($xmldocument && $xsldocument)
    {
      $xmlobj = new DomDocument();
      $xmlobj -> loadxml($xmldocument);
      $xslobj = new DomDocument();
      $xslobj -> loadxml($xsldocument);

      $xsltref = new XSLTProcessor;
      $xsltref -> importStylesheet($xslobj);
      $xsl_output = $xsltref -> transformToXML($xmlobj);

      return $xsl_output;
    }
  }
?>
```

```php
<?php
  // Funktionsaufruf:
  header("Content-type: text/html; charset=UTF-8");
  require_once("transform.php");
  echo XSLTrans();
?>
```

- Demo im Verzeichnis PHP5

# 10. Demonstrationen und Diskussion

## (3) Ausgabe als CSV-Format

- Einfache Form der Datenablage als Schnittstelle zu Anwendungen
- Umsetzung mit XSLT:
  - ✓ Ausgabemethode text
  - ✓ Semikolon als Trennzeichen ($trz)
  - ✓ Zeilenumbrüche mit \n = 
 ($lf)
  - ✓ Kodierung für MS Excel: Windows-1252

```
<xsl:template match="werk">
  <!-- Zeile aufbauen -->
  <xsl:variable name="zeile" select="fn:concat(
    fn:position(), $trz, @jahr, $trz, werkname, $trz,
    if (fn:string-length(@typ) > 0) then @typ else '',
    $trz, fn:count(titelliste/titel), $lf)"/>

  <!-- Zeile ausgeben -->
  <xsl:value-of select="$zeile"/>
</xsl:template>
```

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | 1 | 1981 | Kollaps | LP | 14 |
| 2 | 2 | 1983 | Zeichnungen | LP | 13 |
| 3 | 3 | 1984 | Strategies Aç | LP | 14 |

- Ergebnis 1;1981;Kollaps;LP;14 [= Zeile 1 in der Excel-Ansicht] usw.
- Demo im Verzeichnis CSV

## (4) Ausgabe als SpreadsheetML (Excel ab 2003)

- Vorarbeit: Export und Analyse einer Arbeitsmappe als XML-Kalkulationstabelle
- Umsetzung mit XSLT:
  - ✓ MS Office-Namensräume zuweisen
  - ✓ Element Workbook (= Arbeitsmappe)
  - ✓ Element DocumentProperties (Title, …)
  - ✓ Element Worksheet (= einzelne Tabelle)
  - ✓ Elemente Table / Column / Row / Cell / Data befüllen
  - ✓ Optionale Zuweisung von Style-Informationen

```
<xsl:template match="werk">

  <Row>
    <Cell><Data ss:Type="Number"><xsl:value-of select="fn:position()"/></Data></Cell>
    <Cell><Data ss:Type="Number"><xsl:value-of select="@jahr"/></Data></Cell>
    <Cell><Data ss:Type="String"><xsl:value-of select="werkname"/></Data></Cell>
    <Cell><Data ss:Type="String">
          <xsl:value-of select="if (fn:string-length(@typ) > 0) then @typ else ''"/></Data></Cell>
    <Cell><Data ss:Type="Number"><xsl:value-of select="fn:count(titelliste/titel)"/></Data></Cell>
  </Row>

</xsl:template>
```

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | **Nr.** | **Jahr** | **Werktitel** | **Typ** | **Songs** |
| 2 | 1 | 1981 | Kollaps | LP | 14 |
| 3 | 2 | 1983 | Zeichnungen des Patienten O.T. | LP | 13 |
| 4 | 3 | 1984 | Strategies Against Architecture (1980 – 1983) | LP | 14 |

- Demo im Verzeichnis Excel

# 10. Demonstrationen und Diskussion

## (5) Ausgabe als XHTML

- Generierung einer XHTML 1.0-Ausgabe der kompletten Diskografie
- Umsetzung mit XSLT:
  - ✓ DOCTYPE XHTML 1.0 Strict und XHTML-Namensraum
  - ✓ Grundgerüst (html, head, body)
  - ✓ Typische Struktur- und Inhaltselemente (a, div, em, hx, img, li, ol, p, strong)
  - ✓ Minimales CSS zur Formatierung
  - ✓ Optionale Elemente / Attribute behandeln
  - ✓ Sortierung und Gruppierung der Werktitel nach Jahren
  - ✓ Ermittlung der Gesamtspielzeit (externe Funktion)
  - ✓ Bewertung (1 – 10) in entsprechende Anzahl Sterne umwandeln:

```
<xsl:if test="@bewertung >= 1">
  <xsl:for-each select="1 to @bewertung">
    <img src="{$bildpfad}/stern.gif" alt="*" />
  </xsl:for-each>
</xsl:if>
```

- Demo im Verzeichnis XHTML

# 10. Demonstrationen und Diskussion

## (6) Ausgabe als HTML5

- Diskografie mit HTML5-Erweiterungen (zum DOCTYPE siehe Code)
- Umsetzung mit XSLT:
  - ✓ Vorgehen weitgehend analog zu XHTML
  - ✓ + Verwendung von article, header, section
  - ✓ + Pro Werk ein article-Element
  - ✓ + Bandinfo und Referenzen als article:

```
<article>
  <!-- optionalen Bandinfotext ausgeben -->
  <xsl:apply-templates select="bandinfo[fn:string-length(.) > 0]"/>

  <!-- optionale Referenzen verarbeiten -->
  <xsl:apply-templates select="referenzen"/>
</article>

<xsl:template match="bandinfo">
  <section>
    <header><h2>Einführung</h2></header>
    <p><strong><xsl:value-of select="."/></strong></p>
  </section>
</xsl:template>
```

- Demo im Verzeichnis HTML5

**[13[7]] Ende Neu**

1996 (CD) ★★★★★★★★★★


1. Was Ist Ist (3:29)
2. Stella Maris (5:18)
3. Die Explosion Im Festspielhaus (4:30)
4. Installation No.1 (4:29)
5. NNNAAAMMM (10:59)
6. Ende Neu (4:57)
7. The Garden (5:10)
8. Der Schacht Von Babel (2:46)

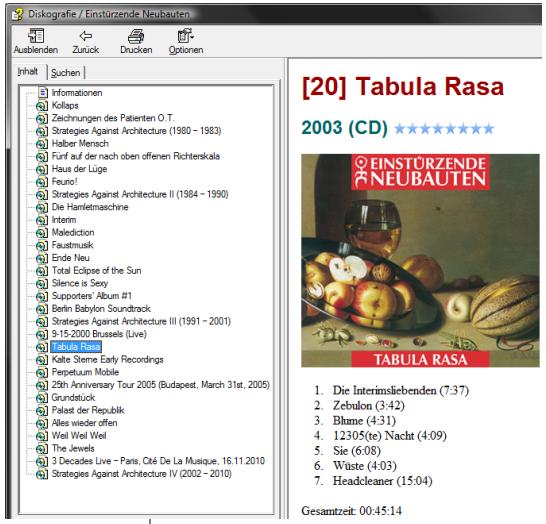
Gesamtzeit: 00:41:38

Kommentar: *Die Lieder wurden aufgenommen*

# 10. Demonstrationen und Diskussion

## (7) Ausgabe als HTML Help

- Diskografie als kompilierte CHM-Datei
- Umsetzung mit XSLT:
  - ✓ Einsatz von xsl:result-document zur Ausgabe von:
    - Projektdatei (.hhp)
    - Inhaltsdatei (.hhc)
    - Externes Stylesheet
    - Topics als einzelne XHTML-Dateien
  - ✓ Kompilierung:
    - HTML Help Workshop:
    - Konsole: hhc ausgabe.hhp




```
<xsl:for-each select="werk">
  <xsl:sort select="@jahr" order="ascending" data-type="number"/>
  <li><object type="text/sitemap">
    <param name="Name" value="{werkname}"/>
    <param name="Local" value="werk_{fn:position()}.html"/>
    <param name="ImageNumber" value="31"/>
  </object></li>
</xsl:for-each>
```

- Demo im Verzeichnis CHM

# 10. Demonstrationen und Diskussion

## (8) Ausgabe als DocBook

- Gesamte Diskografie als Buchstruktur
- Umsetzung mit XSLT:
  - ✓ book-Element (DocBook 5 mit DTD- bzw. RNG-Schemareferenz)
  - ✓ chapter-Elemente für Informationen, Referenzen und Werke
  - ✓ Inhaltselemente: biblioentry, emphasis, imageobject, listitem, orderedlist, para, …

```
<!-- Template für die referenz-Verarbeitung als biblioentry -->
<xsl:template match="referenz">
  <biblioentry>
    <title>
      <xsl:choose>
        <xsl:when test="fn:string-length(@url) > 0">
          <link xlink:href="{@url}"><xsl:value-of select="."/></link>
        </xsl:when>
        <xsl:otherwise>
          <xsl:value-of select="."/>
        </xsl:otherwise>
      </xsl:choose>
    </title>
  </biblioentry>
</xsl:template>
```

- Demo im Verzeichnis DocBook

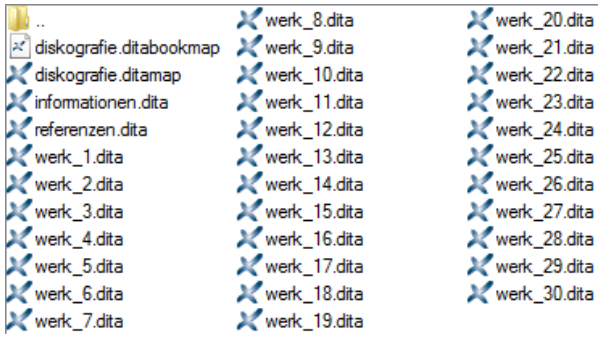
# 10. Demonstrationen und Diskussion

## (9) Ausgabe als DITA

- Topics und Maps
- Umsetzung mit XSLT:
  - ✓ Einsatz von xsl:result-document für Mehrfachausgaben
  - ✓ Topics für Informationen, Referenzen und pro Werk
  - ✓ Topic-Map sowie Book-Map (DITA 1.2)
  - ✓ Template-Steuerung mit mode-Attribut

```
<xsl:output name="dita_topic" method="xml" version="1.0" encoding="UTF-8" indent="yes"
                              doctype-public="..." doctype-system="..."/>


<xsl:template match="werk">
  <xsl:result-document href="{$dokpfad}/werk_{fn:position()}.dita" format="dita_topic">
    <topic id="werk_{fn:position()}">
      <!-- ... -->
    </topic>
  </xsl:result-document>
</xsl:template>
```




- Demo im Verzeichnis DITA

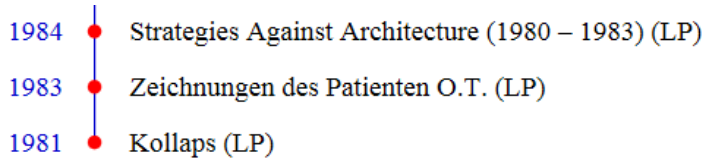
# 10. Demonstrationen und Diskussion

## (10) Ausgabe als SVG

- Idee: Zeitstrahl als Vektorgrafik (Linie, Kreise, Texte)
- Umsetzung mit XSLT:
  - ✓ DOCTYPE für SVG 1.1 und SVG-Namensraum
  - ✓ Konfiguration für flexible Anzeige mittels viewBox
  - ✓ Vertikale Anordnung der Jahreszahlen und Werktitel
  - ✓ Sortierung und Gruppierung der Werktitel nach Jahren
  - ✓ Einsatz von Variablen und Rechenoperationen
  - ✓ Belegung diverser Attribute (Koordinaten) mit AVTs {…}

```
<!-- Jahreszahl + Kreis -->
<xsl:variable name="akt_y" select="$basis_y - (fn:position() - 1) * $delta_px"/>
<text x="20" y="{$akt_y + 5}" fill="#00C"><xsl:value-of select="@jahr"/></text>
<circle r="{$kreis_r}" cx="{$basis_x}" cy="{$akt_y}" fill="#F00"/>

<!-- Werk-Titel (optional Typ), ggf. mehrere getrennt und ××× als Trennung -->
<text x="90" y="{$akt_y + 5}">
  <xsl:for-each select="current-group()">
    <xsl:value-of select="werkname"/>
    <!-- ... -->
  </xsl:for-each>
</text>
```




- Demo im Verzeichnis SVG

# 10. Demonstrationen und Diskussion

## (11) Ausgabe als JSON

- JavaScript Object Notation als alternative Datenablage (siehe json.org)
- Umsetzung mit XSLT:
  - ✓ Ausgabe als text/plain
  - ✓ Hierarchie mit Objekten {…}, Arrays […] und Strings "…" abbilden
  - ✓ Kritische Zeichen in den Zeichenketten mit fn:replace() und RegEx maskieren
  - ✓ Ergebnis kann als JS-Objekt direkt in HTML verarbeitet werden

```
<xsl:template match="diskografie">
  <xsl:text>{"diskografie" : {
  </xsl:text>
  <xsl:text>"bandname" : </xsl:text><xsl:text>"</xsl:text><xsl:value-of select="tm:jsonescape(@bandname)"/><xsl:text>",
  </xsl:text>
  <!-- werk-Verarbeitung -->
  <xsl:text>"werk" : [</xsl:text>
  <xsl:apply-templates select="werk">
    <xsl:sort select="@jahr" order="ascending" data-type="number"/>
  </xsl:apply-templates>
  <xsl:text>]
  }}</xsl:text>
</xsl:template>
```

- Demo im Verzeichnis JSON

# 10. Demonstrationen und Diskussion

## (12) Ausgabe als XSL-FO → PDF

- Druckfähige Diskografie
- Umsetzung mit XSLT:
  - ✓ Generierung der XSL-FO-Struktur mit Titelseite und Seiten pro Werk
  - ✓ Einsatz der spezifischen Elemente aus dem fo-Namensraum
  - ✓ Kopfzeile mit statischem Text und Fußzeile mit Seitenzahl
  - ✓ Umgang mit Seitenumbrüchen und Formatierungen
  - ✓ Typische Elemente zur Ausgabe von Texten, Bildern, Listen
  - ✓ PDF-Tests mit Apache FOP und Antenna House Formatter

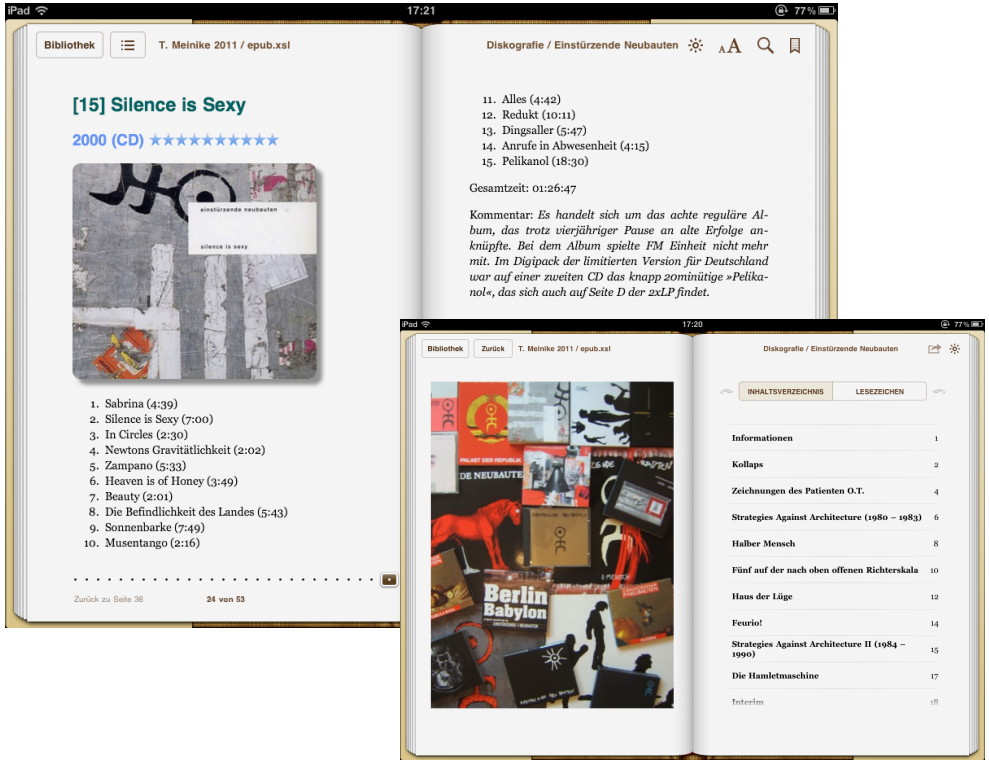
```
<!-- Werktitel als Überschrift ausgeben -->
<fo:block font-family="Arial, Helvetica, sans-serif"
  color="#066" font-weight="bold" font-size="18pt"
  space-after="0.5cm">
  <xsl:value-of select="werkname"/>
</fo:block>
```


- Demo im Verzeichnis FO_PDF

# 10. Demonstrationen und Diskussion

## (13) Ausgabe als EPUB

- Vollständiges E-Book nach EPUB-Standard 2.0.1 (idpf.org)
- Umsetzung mit XSLT:
  - ✓ Steuer- und Inhaltsdokumente über xsl:result-document:
    - mimetype
    - container.xml
    - content.opf
    - toc.ncx
    - style.css
    - *.html (XHTML 1.1)
  - ✓ Batch-Prozess (run.cmd):
    - Vorkopieren von Bildern
    - Transformation mit Saxon
    - Packen mit Info-Zip
    - Prüfung mit EpubCheck
  - ✓ Test mit iBooks & Co.
- Demo im Verzeichnis EPUB



# 10. Demonstrationen und Diskussion

## (14) Ausgabe als XML-Restrukturierung

- Teilumformung der Diskografie

```
<xsl:template match="@* | node()">
  <xsl:copy>
    <xsl:apply-templates select="@* | node()"/>
  </xsl:copy>
</xsl:template>
```

- Umsetzung mit XSLT:
  - ✓ Identitäts-Template als Basis
  - ✓ Zusätzliches id-Attribut bei werk-Elementen anlegen
  - ✓ Attribute @jahr und @typ in Elemente jahr und typ wandeln
  - ✓ Element Kommentar entfernen
  - ✓ XML-Schema diskografie_neu.xsd zuweisen

```
<xsl:template match="werk">
  <xsl:variable name="pos"><xsl:number format="1"/></xsl:variable>
  <xsl:copy>
    <xsl:attribute name="id"><xsl:value-of select="fn:concat('werk_', $pos)"/></xsl:attribute>
    <xsl:for-each select="@*">
      <xsl:element name="{local-name()}"><xsl:value-of select="."/></xsl:element>
    </xsl:for-each>
    <xsl:apply-templates select="node()"/>
  </xsl:copy>
</xsl:template>

<xsl:template match="kommentar"/>
```

- Demo im Verzeichnis Restrukturierung

# Ausblick und Referenzen

**@danielrendall**
Daniel Rendall

Still amazed how ordinary people manage without knowledge of XML and XSLT. Must be like living in the dark ages.

# Ausblick und Referenzen (1)

- Die Stärken von XSLT lassen sich besonders in den Bereichen Technische Dokumentation / Kommunikation, Informationsentwicklung und Content-Management ausnutzen
- XSLT erfordert viel Praxis und konkrete Anwendungen, siehe Beispiele:
  http://www.iks.hs-merseburg.de/~meinike/PDF/tekom2011_OTS11_Meinike_Beispiele.zip
- Motto des Autors: Learning by Coding
- Ein guter Startpunkt für XSLT und XPath 1.0 ist noch immer SELFHTML:
  http://de.selfhtml.org/xml/darstellung/
- W3C-Spezifikationen: http://www.w3.org/Style/XSL/
- tekom-Vortrag zu XSLT / XPath 2.0 (2007):
  http://www.tekom.de/upload/2284/INF_114_Meinike_Vortrag.pdf
- Zum Nachschlagen und für die Insel:
  Michael Kay: XSLT 2.0 and XPath 2.0 Programmer's Reference, 4th Edition,
  Wrox 2008, ISBN: 978-0-470-19274-0
- Verwendete Tweets: http://twitter.com/

@kinsteronline
kinsteronline

Just threw an XSLT book into the recycle bin, felt great

# Ausblick und Referenzen (2)

- 30 Jahre **Einstürzende Neubauten** kann man nicht allein mit Worten und XML beschreiben. Man muss sie sehen und vor allem hören!
- Mehr auf der Bandwebsite: http://neubauten.org/

# </Tutorial>

Danke für Ihr Interesse!

@davidkaae
David Sørensen

Have seen the light with #xslt