XSLT 2.0 im Browser mit Saxon-CE

tekom-Jahrestagung 2013 – Wiesbaden, 06. November

Dr. Thomas Meinike

Hochschule Merseburg | FB Informatik und Kommunikationssysteme
http://www.iks.hs-merseburg.de/~meinike/
thomas.meinike@hs-merseburg.de

XSLT im Web

- Web-Browser ermöglichen Transformationen von XML-Dokumenten über eine darin angegebene Verarbeitungsanweisung der Form:
  `<?xml-stylesheet href="name.xsl" type="text/xsl"?>`
  Dabei wird üblicherweise HTML erzeugt und direkt angezeigt.
  Support-Testcases: http://greenbytes.de/tech/tc/xslt/

- Mittels JavaScript-Objekten lassen sich ebenfalls clientseitige Transformationen ausführen (unterschiedliche Implementierungen).

- Serverseitige Sprachen wie PHP stellen Module für die XML-Verarbeitung mit XSLT bereit.
  Vorteil: Nur der erzeugte HTML-Code gelangt zum Browser.

- Alle genannten Techniken erlauben bisher nur XSLT / XPath 1.0!

Saxon-CE

- Saxon ist einer der populärsten XSLT-Prozessoren, sein Entwickler Michael Kay ist zudem als W3C-Herausgeber und Buchautor bekannt.
- Die CE-Ausgabe (Client Edition) wurde in Version 1.0 bereits 2011 veröffentlicht, zunächst als kommerzielles Produkt pro Domain.
- Im Februar 2013 erschien Version 1.1 unter einer Open-Source-Lizenz (MPL). Diese ist Gegenstand der folgenden Ausführungen.
- Der Code des in Java entwickelten Saxon-Prozessors wurde mit dem Google Web Toolkit nach JavaScript „cross-kompiliert".
- Unterstützt werden XSLT und XPath 2.0 (siehe Vortrag von 2007).

Erweiterungen von XSLT / XPath 2.0

- Mehrfachausgaben (xsl:result-document)
- Komfortable Gruppierungen (xsl:for-each-group)
- Benutzerdefinierte Funktionen (xsl:function)
- Umfangreiche XML-Schema-Datentypen (xs:…)
- Insgesamt mehr als 100 XPath-Funktionen (fn:…)
- Flexible Sequenzen statt Ergebnisbaumfragmenten
- Neue Operatoren, Abfragen mit regulären Ausdrücken, …

Saxon-CE installieren

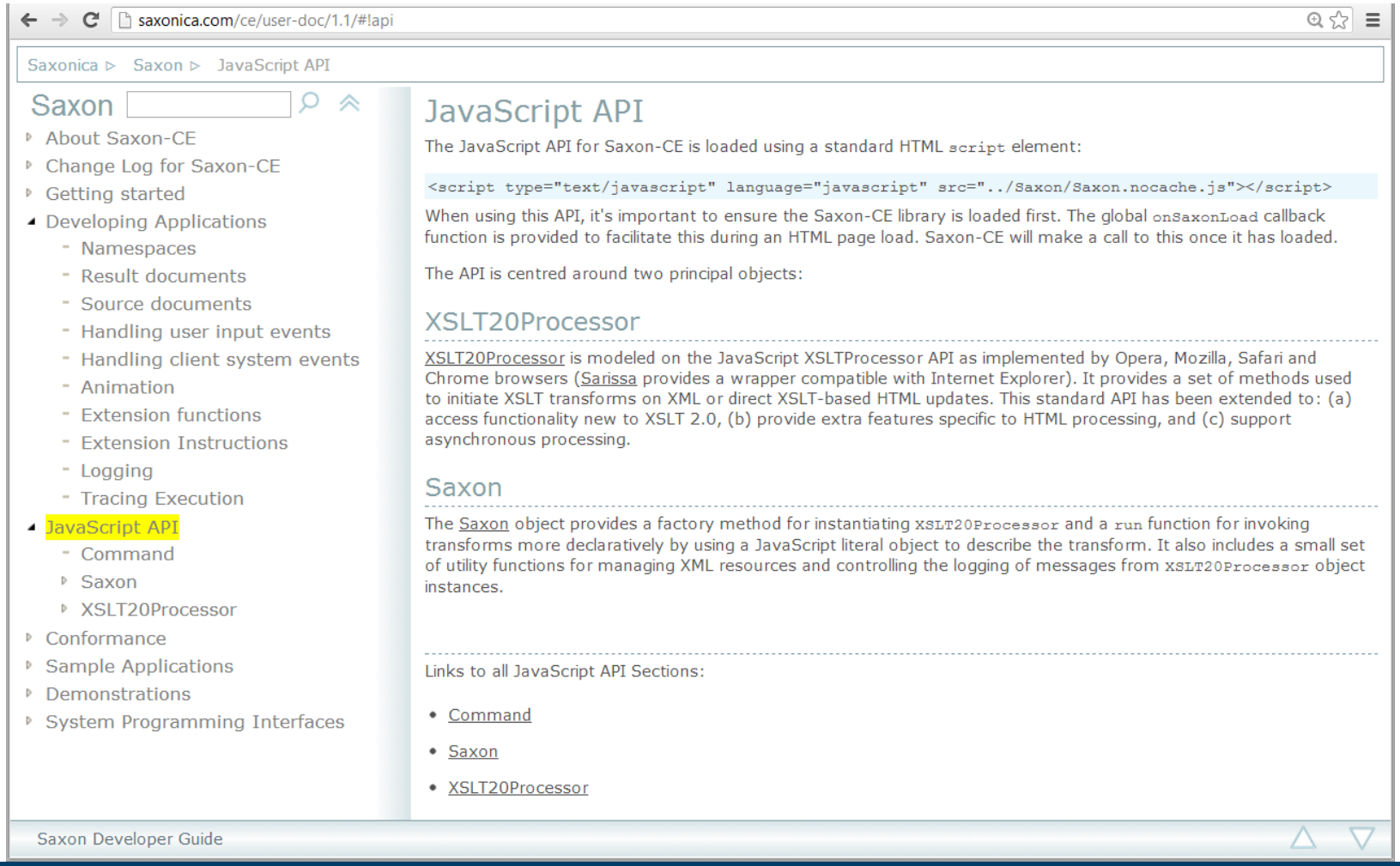
- GitHub-Version *Saxon-CE-master.zip* (5.1 MB) entpacken.
- Enthalten sind die Dokumentation, Beispiele und die Software selbst unter *\releases\1.1\Saxon-CE.1.1.zip*.
- Die Verzeichnisse *Saxonce* bzw. *SaxonceDebug* enthalten die benötigten Komponenten.
- Es wird zunächst das kleine Skript *Saxonce.nocache.js* geladen (< 6 KB), welches Browser-spezifischen Code aus **.cache.html* nachlädt (~ 880 KB).

```
<script type="text/javascript" src="../Saxonce/Saxonce.nocache.js"></script>
```

- Damit kann die Entwicklung beginnen.

CE-Dokumentation online

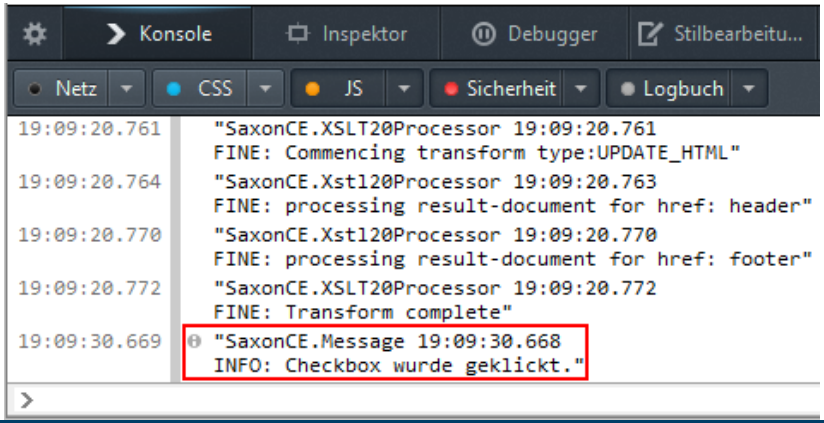
➔ ?...



Allgemeines zur Entwicklung

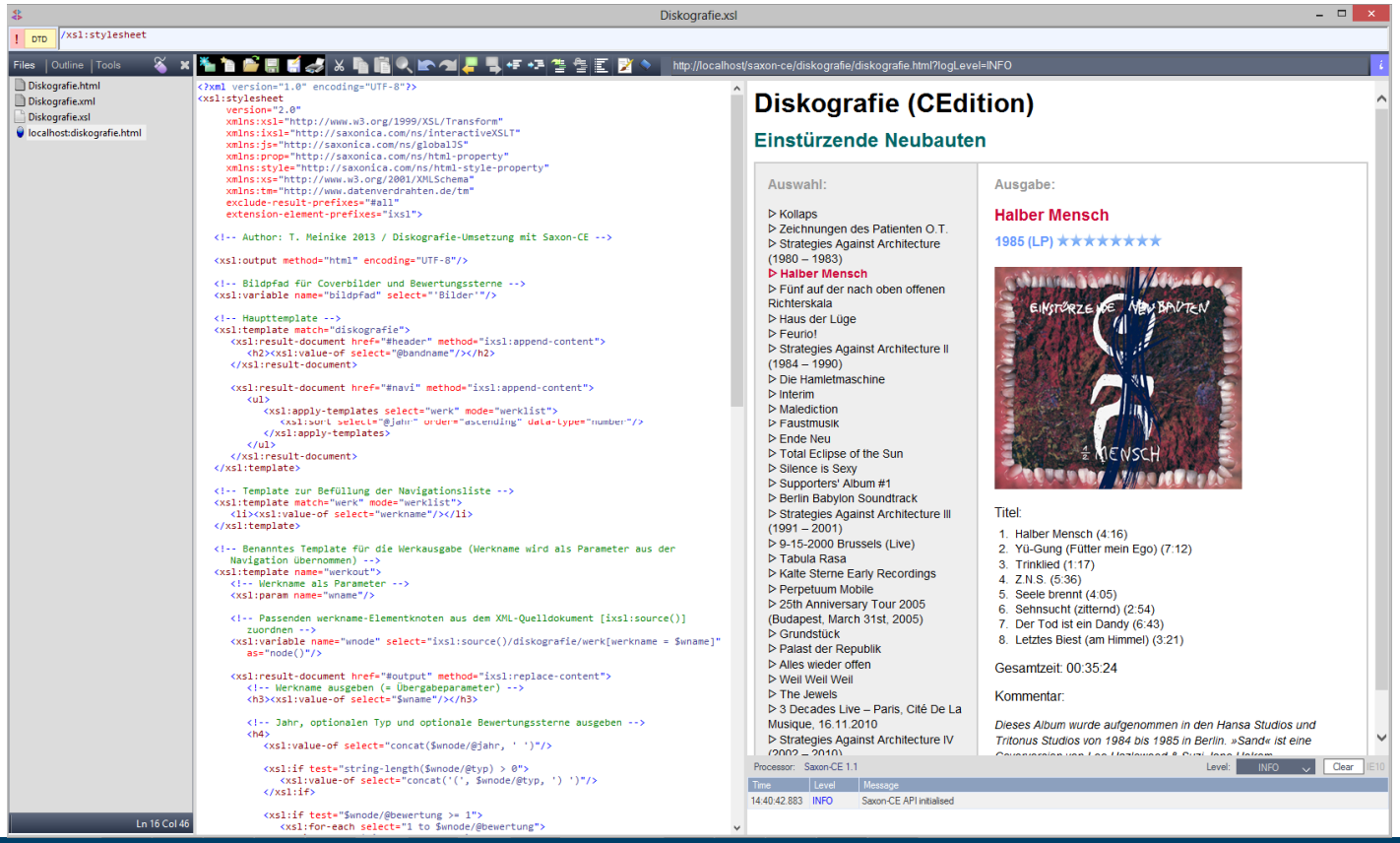
- Nutzung üblicher HTML- / XML-Editoren.
- Code-Ausführung über (lokalen) Web-Server oder direkt im Browser.
- Bei direkter Browsernutzung muss der Zugriff auf externe Ressourcen ggf. explizit erlaubt werden.
- Debugging über die Browserkonsolen (Ctrl + Shift + I bzw. F12 im IE)
  xsl:message liefert Testausgaben:

Mögliche Log-Level:
SEVERE, WARNING, INFO,
FINE, FINER, FINEST



Allgemeines zur Entwicklung

➔ Unter Windows läuft die spezielle CE-Umgebung XMLQuire (benötigt lokalen Webserver, z. B. aus dem XAMPP-Paket):



- Code
- Vorschau
- Fehlerkonsole

Minimalbeispiel 1

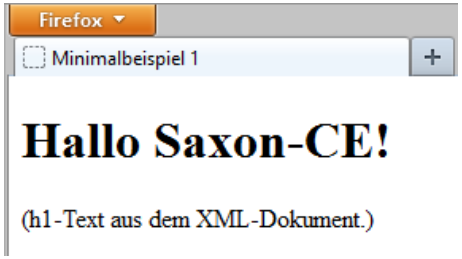
➔ HTML-Code mit JavaScript-Einbindung + XML- und XSLT-Dokument:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" /><title>Minimalbeispiel 1</title>
    <script type="text/javascript" src="../Saxonce/Saxonce.nocache.js"></script>
    <script type="text/javascript"> onSaxonLoad = function() {
      Saxon.run({ stylesheet: "hallo_ce.xsl", source: "hallo_ce.xml", logLevel: "SEVERE" });
    } </script>
  </head>
  <body>
    <!-- Ausgaben landen hier (im DOM) ... -->
  </body>
</html>
```

➔ XML-Code:

```
<?xml version="1.0" encoding="UTF-8"?>
<wurzel>
  <info>Hallo Saxon-CE!</info>
</wurzel>
```

Minimalbeispiel 1



➔ XSLT-Code:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:ixsl="http://saxonica.com/ns/interactiveXSLT"
  xmlns:js="http://saxonica.com/ns/globalJS"
  xmlns:prop="http://saxonica.com/ns/html-property"
  xmlns:style="http://saxonica.com/ns/html-style-property"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  exclude-result-prefixes="#all" extension-element-prefixes="ixsl">
  <xsl:template match="wurzel">
    <h1><xsl:value-of select="info"/></h1><p>(h1-Text aus dem XML-Dokument.)</p>
  </xsl:template>
</xsl:stylesheet>
```

Zusätzliche CE-Namensräume:
ixsl, js, prop, style

Minimalbeispiel 2

➔ Alternative JavaScript-Einbindung nur mit XSLT-Dokument:

```
<script type="text/javascript" src="../Saxonce/Saxonce.nocache.js"></script>
<script type="text/javascript"> onSaxonLoad = function() {
  Saxon.run({ stylesheet: "hallo_ce_2.xsl", initialTemplate: "start", logLevel: "SEVERE" });
} </script>
```

➔ Alternativer XSLT-Code:

```
<xsl:template name="start">
  <h1>Hallo Saxon-CE!</h1>
  <p>(h1-Text im XSLT-Code vorgegeben.)</p>
</xsl:template>
```

Kompaktere Quellen-Einbindung

➔ Minimalbeispiel 1:

```
<script type="text/javascript" src="../Saxonce/Saxonce.nocache.js"></script>

<script type="application/xslt+xml" src="hallo_ce.xsl" data-source="hallo_ce.xml"></script>
```

➔ Minimalbeispiel 2:

```
<script type="text/javascript" src="../Saxonce/Saxonce.nocache.js"></script>

<script type="application/xslt+xml" src="hallo_ce_2.xsl" data-initial-template="start"></script>
```

➔ Log-Parameter bei Bedarf an URL hängen:

```
name.html?logLevel=SEVERE
```

Hinweis:
In der Dokumentation wird für das zweite script-Element noch das zusätzliche Attribut **language="xslt2.0"** angegeben. Dieses führt zur Warnung bei der Validierung (deprecated). Beim Weglassen zeigten sich keine Auswirkungen.

{Demos …}

Spezifische Ausgaben (1)

➔ Beispiel Studierende in Deutschland – Im HTML div-Elemente mit IDs:

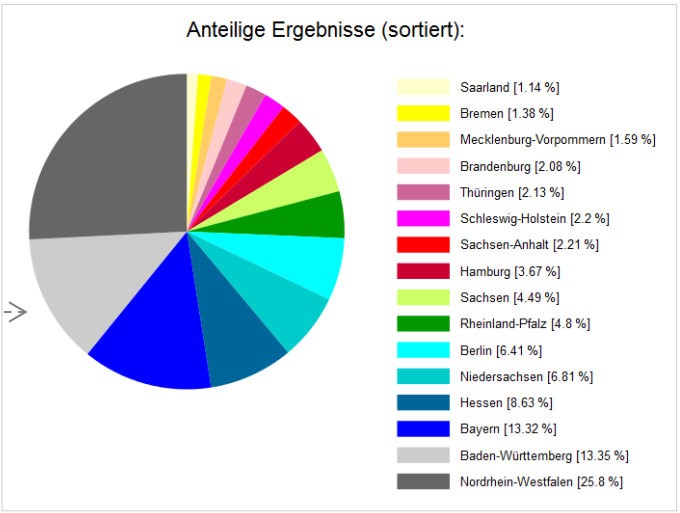
```
<body>
  <div id="header">
    <h1>Studierende</h1>
  </div>
  <div id="ausgabe">
    <div id="tabelle"></div>
    <div id="grafik"></div>
  </div>
  <div id="footer"></div>
</body>
```

Studierende

zum Wintersemester …

○ 2010/2011 ○ 2011/2012 ◉ 2012/2013 ☑ Grafik anzeigen

Bundesland	Semester 2012/2013
Baden-Württemberg	333408
Bayern	332766
Berlin	160145
Brandenburg	51857
Bremen	34383
Hamburg	91546
Hessen	215520
Mecklenburg-Vorpommern	39827
Niedersachsen	170164
Nordrhein-Westfalen	644320
Rheinland-Pfalz	119857
Saarland	28415
Sachsen	112191
Sachsen-Anhalt	55251
Schleswig-Holstein	54935
Thüringen	53234
Gesamt	**2497819**



[Projekt by T. Meinike – 08/2013 | Umgesetzt mit Saxon-CE | Datenquelle: Statistisches Bundesamt]

➔ Ziel: #header, #footer, #tabelle mit HTML und #grafik mit SVG füllen.

Spezifische Ausgaben (2)

- Mit IDs versehende HTML-Elemente lassen sich aus dem XSLT-Kontext mit Inhalten belegen (id="idname").

- Zugriff im XSLT über xsl:result-document und href="#idname":

```
<xsl:result-document href="#idname" method="ixsl:append-content">
  <!-- Ausgaben ... -->
</xsl:result-document>
```

- Das method-Attribut regelt den Umgang mit Inhalten:

  ixsl:append-content … vorhandenen Inhalt ergänzen

  ixsl:replace-content … Inhalt komplett ersetzen

- Der Namensraum ixsl kennzeichnet alle CE-Erweiterungen für **Interactive XSLT** zur Kommunikation mit DOM- und JS-Objekten.

Interactive XSLT (1)

➔ Reaktion auf Benutzerereignisse über bekannte Event-Handler wie onclick, onmouseover, onkeydown usw. in der Form ixsl:onclick:

```
<xsl:template match="th" mode="ixsl:onclick">
  <xsl:result-document href="#tabelle" method="ixsl:replace-content">
    <!-- Code zur Sortierung einer Tabelle beim Anklicken der Kopfzellen ... -->
  </xsl:result-document>
</xsl:template>
```

Siehe Studierende-Beispiel

➔ Attributzuweisung für bereits erzeugte Elemente im DOM ixsl:set-attribute (Entfernen mit ixsl:remove-attribute):

```
<ixsl:set-attribute name="id" select="'akt'"/>
<ixsl:set-attribute name="style:font-weight" select="'bold'"/>
```

Siehe Diskografie-Beispiel

style-Namensraum

Interactive XSLT (2)

➔ Je nach Template-Kontext (hier ohne match-Bezug) ist ixsl:source() nützlich für übergreifende XML-Zugriffe:

```
<xsl:template name="grafik">
  <!-- ... -->
  <xsl:result-document href="#grafik" method="ixsl:replace-content">
    <!-- ... -->
    <!-- Summe der Studierenden nach Semester -->
    <xsl:variable name="summe"
      select="sum(ixsl:source()//bundesland/daten/wert[$sem_index])"/>
    <!-- ... -->
  </xsl:result-document>
</xsl:template>
```

Siehe Studierende-Beispiel

Interactive XSLT (3)

➔ Paralleler Zugriff auf den erzeugten DOM-Baum mittels ixsl:page():

```
<xsl:template match="li" mode="ixsl:onclick">
  <xsl:for-each select="ixsl:page()//div[@id = 'navi']/ul/li">
    <!-- vorher aktives li-Element mit ID suchen und diese entfernen -->
    <ixsl:set-attribute name="id" select="''"/>
  </xsl:for-each>
  <!-- Aktuellen Navigationslisteneintrag hervorheben via li#akt im CSS -->
  <ixsl:set-attribute name="id" select="'akt'"/>

  <!-- Inhalte zum aktuell gewählten Werk transformieren -->
  <xsl:call-template name="werkout">
    <xsl:with-param name="wname" select="."/>
  </xsl:call-template>
</xsl:template>
```

Siehe Diskografie-Beispiel

Interactive XSLT (4)

➔ Ansprechen von JS-/CSS-Eigenschaften mit ixsl:property / @prop:

```
<!-- Grafik aus-/einblenden -->
<xsl:template match="input[@type eq 'checkbox'][@id eq 'gronoff']" mode="ixsl:onclick">
  <ixsl:set-property object="id('grafik')" name="style.visibility"
    select="if (@prop:checked eq 'false') then 'hidden' else 'visible'"/>
</xsl:template>
```

prop-Namensraum

Entspricht dieser JS-DOM-Syntax:
document.getElementById('grafik').style.visibility = 'hidden'; // bzw. 'visible'

Siehe Studierende-Beispiel

Interactive XSLT (5)

➔ Erweiterungen für den JS-Objektzugriff:

Zugriff auf das window-Objekt des Browsers – ixsl:window():

Zugriff auf Objekt-Eigenschaften – ixsl:get():

Ausführung von Objekt-Methoden – ixsl:call():

```
<!-- User-Agent-String des Browsers -->
<xsl:value-of select="ixsl:get(ixsl:get(ixsl:window(), 'navigator'), 'userAgent')"/>

<!-- Sinus-Funktionswert von 1 -->
<xsl:value-of select="ixsl:call(ixsl:get(ixsl:window(), 'Math'), 'sin', 1)"/>
```

Interactive XSLT (6)

➔ Erweiterungen für den JS-Objektzugriff:

Ereignisse abfangen und nutzen – ixsl:event():

```
<xsl:template match="input[@id = 'eingabe']" mode="ixsl:onkeydown">
  <xsl:variable name="event" select="ixsl:event()"/> <!-- object KeyboardEvent -->
  <xsl:variable name="keycode" select="ixsl:get($event, 'keyCode')"/>
  <!-- $keycode weiter verarbeiten, z. B. Pfeiltasten: → 37 ↑ 38 → 39 ↓ 40 -->
</xsl:template>
```

Zeichenketten mit Code auswerten – ixsl:eval():

```
<xsl:value-of select="ixsl:eval('(1 + 2) * 3')"/> <!-- 9 -->
```

Interactive XSLT (7)

➔ Zugriff auf eigene JS-Funktionen über js:myfunction(…):

Im Studierenden-Beispiel werden Sinus- und Cosinus für die SVG-Kreissegmente (Element path) benötigt. Statt mit ixsl:call() wurden die Funktionen in einem externen Skript (zusatz.js) deklariert:

```
function sin(arg) { return Math.sin(arg); }
function cos(arg) { return Math.cos(arg); }
function pi()     { return Math.PI; }
```

Aufruf im XSLT-Code analog zu XPath-Funktionen mit js-Präfix:

```
<xsl:value-of select="js:sin(1)"/> <!-- einfache Ergebnisausgabe -->
<xsl:variable name="punkt_xs" select="round-half-to-even(js:cos($startwinkel *
  js:pi() div 180) * $kreis_r + $kreis_x, 2)"/> <!-- Koordinaten-Berechnung -->
```

Interactive XSLT (8)

➔ ixsl:schedule-action zur verzögerten Ausführung benannter Templates, etwa für zeitlich animierte Ausgaben:

```
<ixsl:schedule-action wait="..."> <!-- wait = Zahlenwert in ms -->
  <xsl:call-template name="...">
    <xsl:with-param name="..." select="..."/> <!-- z. B. @prop:value aus einem Formular -->
  </xsl:call-template>
</ixsl:schedule-action>
```

Der einzige erlaubte Kindinhalt von ixsl:schedule-action ist xsl:call-template zum Aufrufen benannter Templates!

Zusätzliche Techniken (1)

➔ Das Saxon-Objekt stellt neben run(…) weitere Methoden für skriptgesteuerte Transformationen zur Verfügung, u. a. requestXML("…") und newXSLT20Processor(xsltobj):

```
var onSaxonLoad = function()
{
  // Initiale Transformation zur Katalogauswertung:

  var xmldok = Saxon.requestXML("katalog.xml");

  var xsldok = Saxon.requestXML("katalog.xsl");

  var xsltpr = Saxon.newXSLT20Processor(xsldok);

  xsltpr.updateHTMLDocument(xmldok);
};
```

Diskografie deluxe:

- Initiale Transformation verarbeitet katalog.xml und generiert Buttons mit Verweis auf weitere JS-Funktion `SaxonUpdate(xmlfile, bildpfad) {…}`
- Diese Funktion übernimmt XML-Dateiname und Bildpfad, wobei letzterer als Parameter an die folgende Transformation übergeben wird.

Beispielausgabe:

Zusätzliche Techniken (2)

➔ katalog.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE katalog [...]>
<katalog>
  <band>
    <name>Bandname</name>
    <daten>Daten/Bandname.xml</daten>
    <bilder>Bilder/Bandname</bilder>
  </band>
  <band> <!-- Band 2 --> </band>
  <band> <!-- Band 3 --> </band>
</katalog>
```

katalog.xsl:

```
<xsl:template match="katalog">
  <xsl:result-document
    href="#header" method="ixsl:append-content">
    <form name="auswahl">
      <span>Katalogeinträge: </span>
      <xsl:for-each select="band">
        <xsl:sort select="name"
          order="ascending" data-type="text"/>
          <input type="button" value="{name}"
            onclick="SaxonUpdate('{daten}','{bilder}')"/>
      </xsl:for-each>
    </form></xsl:result-document>
</xsl:template>
```

Zusätzliche Techniken (3)

➔ Transformation über die erzeugten Buttons via onclick:

```
function SaxonUpdate(xmlfile, bildpfad)
{
  // Vorherige Ausgaben entfernen (h2-, p-, und div-Elemente)
  // ...
  // Dynamische Transformation:
  var xslfile = "diskodeluxe.xsl";
  var xmldok  = Saxon.requestXML(xmlfile);
  var xsldok  = Saxon.requestXML(xslfile);
  var xsltpr  = Saxon.newXSLT20Processor(xsldok);
  xsltpr.setParameter(null, "bildpfad", bildpfad);
  xsltpr.updateHTMLDocument(xmldok);
}
```

Weitere Details unter:

http://www.saxonica.com/ce/user-doc/1.1/#!api/xslt20processor

Erzeugte Ausgaben analog zur einfachen Diskografie.

{Demos …}

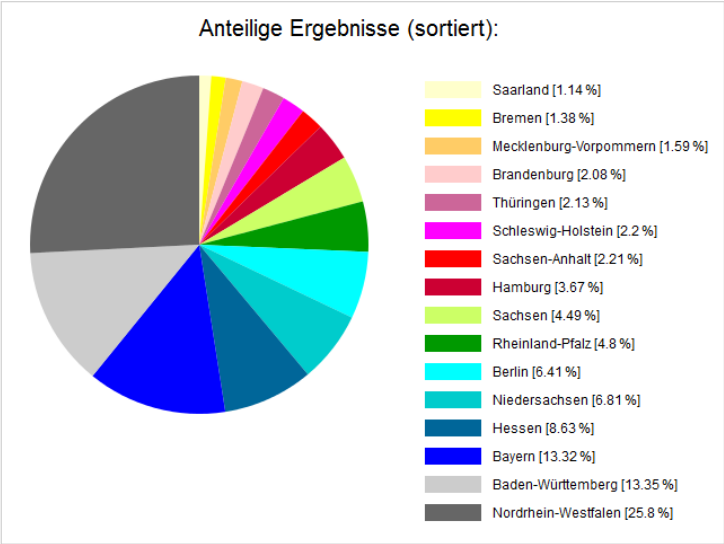
Zum Studierenden-Beispiel

Studierende

zum Wintersemester …

○ 2010/2011 ○ 2011/2012 ◉ 2012/2013 ☑ Grafik anzeigen

Bundesland	Semester 2012/2013
Baden-Württemberg	333408
Bayern	332766
Berlin	160145
Brandenburg	51857
Bremen	34383
Hamburg	91546
Hessen	215520
Mecklenburg-Vorpommern	39827
Niedersachsen	170164
Nordrhein-Westfalen	644320
Rheinland-Pfalz	119857
Saarland	28415
Sachsen	112191
Sachsen-Anhalt	55251
Schleswig-Holstein	54935
Thüringen	53234
Gesamt	**2497819**



[Projekt by T. Meinike – 08/2013 | Umgesetzt mit Saxon-CE | Datenquelle: Statistisches Bundesamt]

Wesentliche Schritte:

- Vorhandene Datensätze abfragen und Radio-Buttons + Grafik-Checkbox anzeigen.
- Nach Anklicken eines Buttons Ausgaben für Tabelle und Grafik erzeugen.
- Tabelle kann dynamisch sortiert werden, die Grafik reagiert auf hover-Events.
- Inhalt im Footer-Bereich mit Link zur Datenquelle aus dem XML-Dokument setzen.
- Mittels Checkbox kann die Grafik aus- bzw. eingeblendet werden.

Zum Diskografie-Beispiel

Diskografie (CEdition)

Einstürzende Neubauten

Bandinfo: Einstürzende Neubauten ist eine deutsche experimentelle Band. Die Berliner Band wurde 1980 von Christian Emmerich), N. U. Unruh (bürgerlich Andrew Chudy), Gudrun Gut und Beate Bartel gegründet; letzte später Mania D. Die Besetzung fluktuierte anfangs und konsolidierte sich 1981 personell um Bargeld, Unruh Band Abwärts hinzugekommenen FM Einheit und Mark Chung sowie den Berliner Underground-Star Alexander H

Referenzen: Eigene Musiksammlung | Offizielle Band-Website | Wikipedia (de) | Wikipedia (en) | indiepedia

Auswahl:

- ▷ Kollaps
- ▷ Zeichnungen des Patienten O.T.
- ▷ Strategies Against Architecture (1980 – 1983)
- ▷ Halber Mensch
- ▷ **Fünf auf der nach oben offenen Richterskala**
- ▷ Haus der Lüge
- ▷ Feurio!
- ▷ Strategies Against Architecture II (1984 – 1990)
- ▷ Die Hamletmaschine
- ▷ Interim
- ▷ Malediction
- ▷ Faustmusik
- ▷ Ende Neu
- ▷ Total Eclipse of the Sun
- ▷ Silence is Sexy
- ▷ Supporters' Album #1
- ▷ Berlin Babylon Soundtrack
- ▷ Strategies Against Architecture III (1991 – 2001)
- ▷ 9-15-2000 Brussels (Live)

Ausgabe:

Fünf auf der nach oben offenen Richterskala

1987 (LP) ★★★★★★★★★

Titel:

1. Zerstörte Zelle (8:01)
2. Morning Dew (4:57)
3. Ich bin's (3:22)
4. Mo Di Mi Do Fr Sa So (4:50)

Diskografie (CEdition)

Katalogeinträge: Einstürzende Neubauten | Nick Cave & The Bad Seeds | Phillip Boa and the Voodooclub

Nick Cave & The Bad Seeds

Bandinfo: The Bad Seeds ist eine australische Rockband, die 1983 vom Sänger Nick Cave, Multiinstrumentalist Mick Harvey und Gitarrist Blixa Bargeld nach Auflösung von The Birthday Party in West-Berlin gegründet wurde. Der Name basiert auf dem Roman The Bad Seed (deutsch: „Böse Saat") von William March. (Wikipedia)

Referenzen: Eigene Musiksammlung | Offizielle Band-Website | Wikipedia (de) | Wikipedia (en) | indiepedia

Wesentliche Schritte:

- Ausgabe der Informationen und Referenzen zur Band.
- Generierung einer ul-Liste mit den Werken (#auswahl).
- Beim Anklicken der Listen-einträge wird der relevante XML-Inhalt abgefragt und ausgeben (#ausgabe).
- Im Code Beachtung, welche XML-Inhalte optional sind (u. a. Kommentar, Zeiten und Bewertung).
- Gesamtspielzeit ermitteln.

Fazit und Ausblick

- Saxon-CE bietet eine interessante Alternative zur Entwicklung von interaktiven Browser-Anwendungen mit beachtlichem Funktions-umfang.

- Das Konzept eignet sich besonders für Online-Hilfesysteme und datengetriebene Dokumentationsprozesse, siehe CE-Dokumentation.

- Dem Produkt ist eine kontinuierliche Weiterentwicklung und kreative Nutzung zu wünschen.

- Danke für Ihr Interesse und viel Erfolg bei der Umsetzung eigener Ideen!

Referenzen

- Saxonica: http://saxonica.com/ce/index.xml
  http://saxonica.com/ce/user-doc/1.1/
- GitHub: https://github.com/Saxonica/Saxon-CE

- XMLQuire Web Edition: http://qutoric.com/xmlquire/ce/

- Meinike, T.:
  - ✔ XSLTuning für Browser; Entwickler Magazin 6.2013, S. 73 – 77
  - ✔ Material zum Projekt Banddiskografie;
    http://www.iks.hs-merseburg.de/~meinike/vortraege.php
  - ✔ XSLT 2.0 und XPath 2.0 für Praktiker, tekom-Jahrestagung 2007;
    http://www.tekom.de/upload/2284/INF_114_Meinike_Vortrag.pdf
  - ✔ Studierende in Deutschland; http://datenverdrahten.de/xslt2/saxon-ce/studis/