

Die wundervolle Welt des XSLT: Outlook

Thomas Meinike
Hochschule Merseburg
thomas.meinike@hs-merseburg.de

Winter School: Digitale Editionen – Masterclass. Wuppertal, 23.02.2021

Themen

- Vorstellung
- Voraussetzungen / Code-Techniken und Konventionen
- Das eigentliche Programm:
 - Erstellung eigener Funktionen mit `xsl:function` und Nutzung vorgefertigter `math:`Funktionen im Kontext von Vektorgrafiken (SVG)
 - Ansätze für Datums-/Zeitoperationen mit `fn:current-date()` / `fn:format-date()` usw.
 - Zugriff auf JSON-Daten mit `fn:json-doc()` & Co.
 - XSLT im Browser mit Saxon-JS
 - Motto: Hands-on Code! Von daher wird viel an Codebeispielen diskutiert und nicht auf alle grundlegenden XSLT-Elemente bzw. XPath-Funktionen auf den Folien eingegangen

Vorstellung

- Lehrkraft für besondere Aufgaben an der Hochschule Merseburg seit 1997:
 - Grundlagen der Online-Dokumentation, Web-Entwicklung und XML-Technologien
- Tätig in den Studiengängen:
 - Bachelor Technisches Informationsdesign
 - Master Informationsdesign und Medienmanagement
- Lehr- und Arbeitsgebiete:
 - Auszeichnungssprachen
 - Content-Management
 - Online-Hilfen und E-Books
 - Web-Entwicklung
 - XML-Technologien
- <https://datenverdrahten.de/> | <https://speakerdeck.com/xmlarbyter> | <https://twitter.com/XMLArbyter>



Voraussetzungen ^{1/2}

- Codebasis liegt komplett vor unter <https://datenverdrahten.de/xslt3/wsde21/xsltoutlook.zip>
- Einsatz des <Oxygen/> XML Editors | `<xsl:stylesheet version="3.0" ...>...</xsl:stylesheet>`
- Codeausführung weitgehend mit dem Prozessor Saxon-HE, der in 10.x sehr viel XSLT 3.0 und XPath 3.1 unterstützt, aber auch der aktuell integrierte 9.9.x bietet für diesen Kurs fast alles an
- Im Code sind gängige Namensräume (`xmlns:name="..."`) an den Anfang gestellt:
(array, fn, map, math, xs, xsl, ...), unabhängig von der jeweiligen konkreten Nutzung
- `expand-text="yes"` ermöglicht ab 3.0 Text Value Templates `{...}`, z. B. `<p>{...}</p>`, analog zu seit 1.0 bekannten Attribute Value Templates `...` und erspart somit `<xsl:value-of select="...">`
- Direkte XSLT-Codetests ohne Zugriff auf XML-Daten lassen sich so realisieren:
`<xsl:template name="xsl:initial-template">...</xsl:template>`
- Lokaler Webserver für Saxon-JS (XAMPP → <https://apachefriends.org/> | Daten unter htdocs)

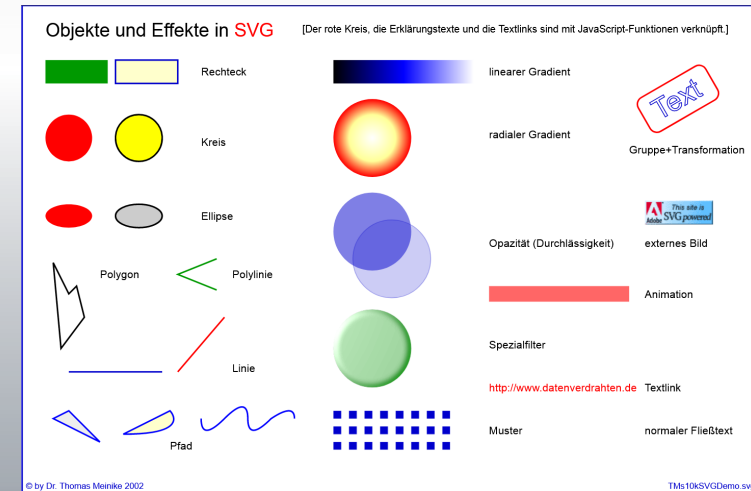
Voraussetzungen ^{2/2}

- Grundlegende Techniken:
 - `xsl:template` / `xsl:apply-templates` / `xsl:for-each`
 - `xsl:if`, `xsl:choose` ... `xsl:when` ... `xsl:otherwise`
 - `xsl:sort` (unter `xsl:apply-templates` / `xsl:for-each` / `xsl:for-each-group`)
 - `xsl:variable` / `xsl:param` → `$name`
- Schema-Datentypen (ab 2.0):
 - `xs:date`, `xs:decimal`, `xs:double`, `xs:integer`, `xs:string`, ...
- Sequenzen als allgemeine Container für iterierbare Knoten-Listen oder atomare Werte (z. B. Zahlenwerte, Zeichenketten)
- Evtl. Vorkenntnisse zu SVG (auch HTML / CSS hilfreich):
 - Grafische Grundformen wie `circle`, `rect`, `path`, ... und Eigenschaften wie `fill`, `stroke`, ...

(: SVG :)

- **Vektorgrafikstandard vom W3C seit 2001, XML-basiert (also auch mit XSLT zugänglich)**
 - Grafische Grundformen wie circle, ellipse, **line**, rect, **path**, polygon, **polyline**
 - Format-Attribute / CSS-Eigenschaften wie **fill**, **stroke**, opacity, ...
 - Animationen, Filter, Gradienten, Pattern, Symbole, Transformationen, ...
- **Objekte und Effekte (die Ur-Demo von 2002):**
 - <https://datenverdrahten.de/svglbc/svg/TMs10kSVGDemo.htm>
 - Hovern über die Infotexte zeigt unten die Codebasis mit den jeweiligen Koordinaten
- **Grundgerüst:**

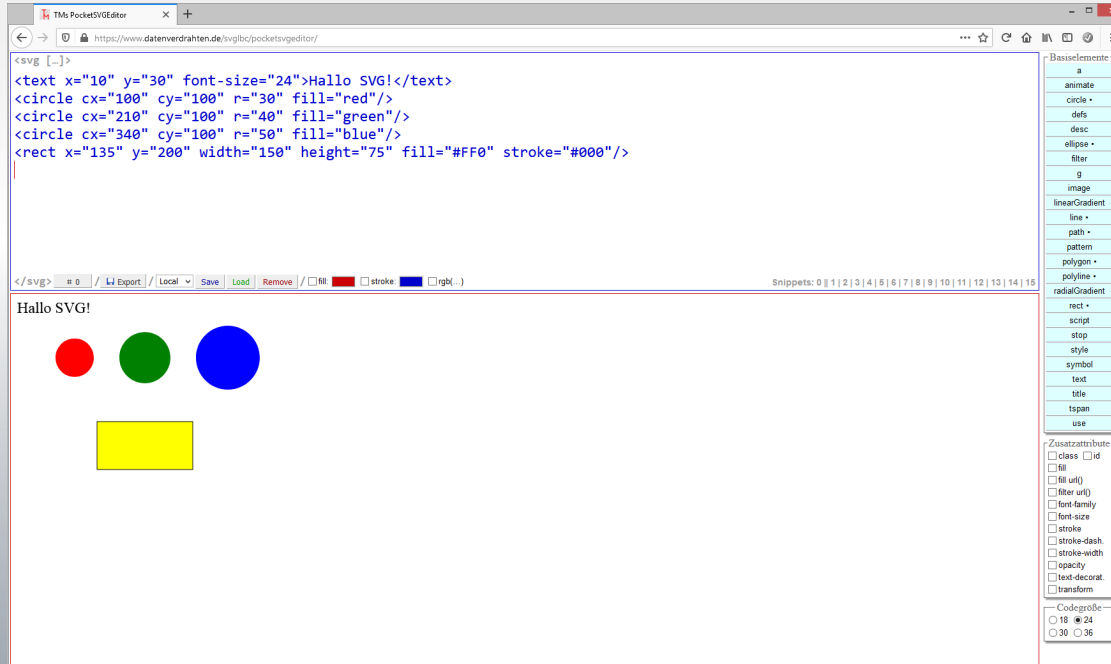
```
<?xml version="1.0" encoding="UTF-8"?>
<svg xmlns="http://www.w3.org/2000/svg">
  <!-- ... -->
</svg>
```



SVG 2/5

- SVG-Lernumgebung aka „Spielwiese“

- <https://datenverdrahten.de/svglbc/pocketsvgeditor/>



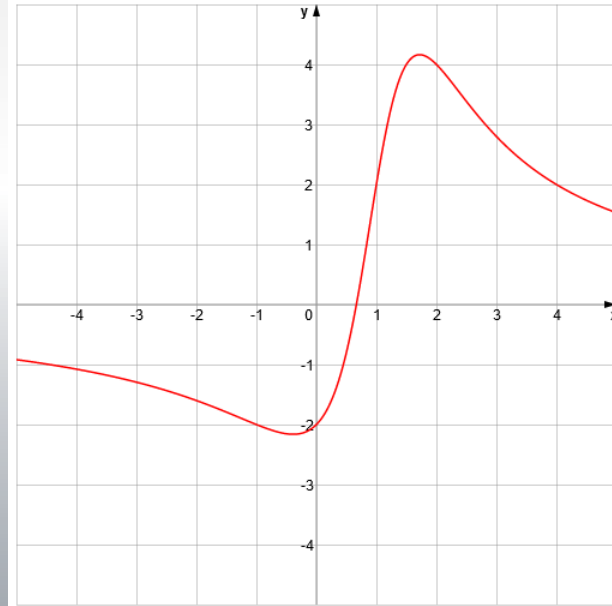
- **Beispiel: Funktionsplot als <polyline points="x1,y1 x2,y2 ... xn,yn"/>**

- xsl:function-Basis:
 - Unterhalb von xsl:stylesheet
 - Funktionsname im name-Attribut mit eigenem Namespace-Präfix, z. B. my (tm im Beispiel)
 - Argument(e) via xsl:param + Datentyp
 - Rückgabe-Datentyp bei xsl:function angeben
 - Nutzung im XSLT-Code als my:function(arg1, arg2, ...)
- Funktion 1 (von 5):

```
<!-- Funktion 1:  $y = (6x - 4) / (x^2 - 2x + 2)$  -->
<xsl:function name="tm:fkt1" as="xs:double">
  <xsl:param name="x" as="xs:double"/>
  <xsl:sequence select="(6 * $x - 4) div
    (math:pow($x, 2) - 2 * $x + 2)"/>
</xsl:function>
```

```
<polyline class="farbe1">
  <xsl:attribute name="points">
    <xsl:for-each select="...">
      <!-- x,y-Berechnung mit Fkt ... -->
    </xsl:for-each>
  </xsl:attribute>
</polyline>
```

Funktionen zeichnen mit SVG

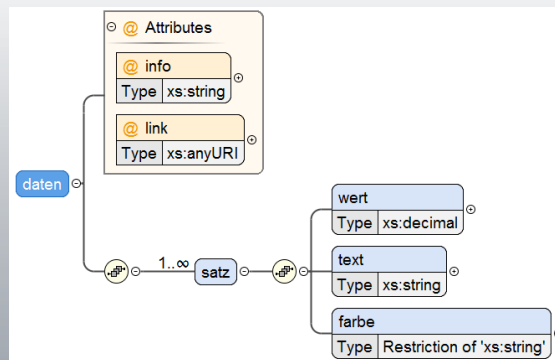


- **Beispiel: Kreisdiagramm (Pie Chart)**

- Datengrundlage: <https://gs.statcounter.com/os-market-share/all/germany/>



- XML-Datensatz erzeugt (XML-Schema-gestützt):



```

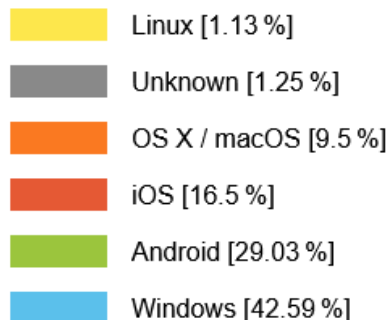
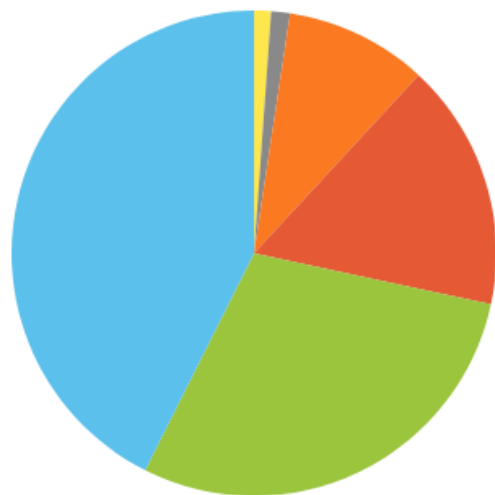
<?xml version="1.0" encoding="UTF-8"?>
<daten ... info="..." link="...">
  <satz>
    <wert>29.03</wert>
    <text>Android</text>
    <farbe>#9BC53D</farbe>
  </satz>
  <!-- 4 weitere satz-Elemente ... -->
  <satz>
    <wert>42.59</wert>
    <text>Windows</text>
    <farbe>#5BC0EB</farbe>
  </satz>
</daten>

```

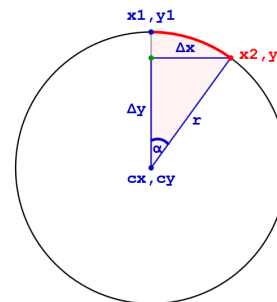
- **Beispiel: Kreisdiagramm (Pie Chart)**

- XML-Transformation nach SVG unter Nutzung der Funktionen `math:sin()` und `math:cos()`
- Ergebnis und Details der Pfadsegment-Berechnung (path)

Operating System Market Share in Germany – January 2021



Kreissegmente zeichnen / berechnen



$$x2 = cx + \Delta x = cx + r \cdot \sin \alpha$$

$$y2 = cy - \Delta y = cy - r \cdot \cos \alpha$$

Kreissegment $\leq 180^\circ$
`<path d="Mcx,cy Lx1,y1 Ar,r 0 0,1 x2,y2 Z" ... />`
 Kreissegment $> 180^\circ$
`<path d="Mcx,cy Lx1,y1 Ar,r 0 1,1 x2,y2 Z" ... />`

(: Datum und Zeit :)

Datum und Zeit_{1/3}

- **Ab XSLT 2.0 stehen Funktionen für Datum und Zeit sowie Formatierungen zur Verfügung**
 - Abfragen:
 - `fn:current-date()` = Datum wie 2021-02-20+01:00
 - `fn:current-time()` = Zeit wie 15:04:04.064+01:00
 - `fn:current-dateTime()` = Datum + Zeit wie 2021-02-20T15:04:04.064+01:00
 - Formatierungen über sog. „Picture-String“ mit [...]-Blöcken für Details:
 - `fn:format-date()`
 - `fn:format-time()`
 - `fn:format-dateTime()`
 - Beispiele:
 - `fn:format-date(fn:current-date(), '[D01].[M01].[Y0001]')` → 20.02.2021
 - `fn:format-time(fn:current-time(), '[H01]:[m01]')` → 15:04

Datum und Zeit _{2/3}

- **Weitere Operationen (siehe Code)**

- Datum erzeugen:

`xs:date('YYYY-MM-DD')`

- Tage zwischen Datumswerten:

`fn:days-from-duration($enddatum – $startdatum)`

- Datum in x Tagen:

`$startdatum + x * xs:dayTimeDuration('P1D')` mit P = Period, hier 1 Day

Datum und Zeit_{3/3}

- **Beispiel: An welchen Tagen liegen die #DHAL-Stammtische?**

- Datum mit 1. eines Monats erzeugen, z. B. `xs:date('2021-02-01')`
- Mit [F] den Wochentag ermitteln (Monday)
- Über `xsl:choose` den folgenden Dienstag finden, 21 Tage addieren und neues Datum auslesen:

#DHAL-Termine

Ab Januar 2021: regulär jeden vierten Dienstag im Monat um 19:30 Uhr

Der 1. im Monat: **Monday**

1. Dienstag: **02.02.2021**

4. Dienstag: **23.02.2021**

- Zu einer Funktion für Jahrestermine erweitern

→ <https://torstenroeder.github.io/dhal/>

#DHAL - Hallenser Digital Humanities Stammtisch

ab Januar 2021: regulär jeden vierten Dienstag im Monat um 19:30

~ab September 2020: regulär jeden vierten Dienstag im Monat um 19:00~

~regulär jeden Dienstag in den durch drei teilbaren Kalenderwochen~

Infokanäle

- >> Mailingliste DH Halle (dh@informatik.uni-halle.de, Anmeldung [hier](#))
- >> Mailingliste DH Mitteldeutschland (dhmdl@lists.uni-leipzig.de)
- >> Mailingliste Forum DH Leipzig (fdhl@lists.uni-leipzig.de)
- >> Twitter: Hashtag [#DHAL](#)
- >> Telegram: Gruppe #DHAL (bitte Kontakt mit einem Gruppenmitglied aufnehmen)
- >> Discord-Server: ~DHAL~ DHall, Channel [#halle_dhal](#)

Nächste Termine - jeweils ab 19.30 Uhr

26. Jan #DHAL 39 virtuDHAL auf dem Discord-Server #DHall, Channel [halle_dhal](#)
23. Feb #DHAL 40 virtuDHAL auf dem Discord-Server #DHall, Channel [halle_dhal](#)
23. Mär #DHAL 41 virtuDHAL auf dem Discord-Server #DHall, Channel [halle_dhal](#)

1. 26.01.2021
2. 23.02.2021
3. 23.03.2021
4. 27.04.2021
5. 25.05.2021
6. 22.06.2021
7. 27.07.2021
8. 24.08.2021
9. 28.09.2021
10. 26.10.2021
11. 23.11.2021
12. 28.12.2021

(: JSON :)

JSON _{1/3}

- **XSLT 3.0 / XPath 3.1 können neben XML auch JSON-Daten verarbeiten**
 - Hier kommen weitere neue Konzepte wie Arrays [...] und Maps {...} ins Spiel
- JSON ist die JavaScript Object Notation, 2001 von Douglas Crockford entworfen und Standard ECMA-404 „The JSON Data Interchange Syntax“ → <https://json.org/>
- Datenkodierung mit den aus JavaScript bekannten Konstrukten:
 - Object {...}
 - Array [...]
 - Zeichenketten in Anführungszeichen (String)
 - Zahlenwerte als ganze oder Fließkommazahlen (Number)
 - Wahrheitswerte true bzw. false (Boolean) sowie null (Nullwert)
- Datenzuweisung in Form von Key-Value-Paaren: **"key": "value"**

JSON _{2/3}

- Beispiel: Bücherliste (buecher.json)

```
{
  "buecher": {
    "info": "Bücherliste",
    "waehrung": "EUR",
    "buch": [
      {
        "nr": 1,
        "autor_in": "Sal Mangano",
        "titel": "XSLT Kochbuch",
        "verlag": "O'REILLY",
        "jahr": 2006,
        "preis": 52.00,
        "isbn": "978-3-89721-457-6"
      },
      2 weitere Einträge ...
    ]
  }
}
```

```
<xsl:variable name="json" select="fn:json-doc('buecher.json')" as="map(*)"/>
<xsl:variable name="buecher" select="$json?buecher" as="map(*)"/>
<xsl:variable name="buch" select="$json?buecher?buch" as="array(*)"/>
<xsl:variable name="waehrung" select="$buecher?waehrung" as="xs:string"/>
<xsl:variable name="buchanzahl" select="array:size($buch)" as="xs:integer"/>
```

```
<p>Infotext: <strong>{$buecher?info}</strong></p>
<p>Währung: <strong>{$waehrung}</strong></p>
<p>Bücheranzahl: <strong>{$buchanzahl}</strong></p>
<p>Preis vom 1. Buch: <strong>{$buch(1)?preis}</strong></p>
<p>Autor*in vom 2. Buch: <strong>{$buch(2)?autor_in}</strong></p>
<p>Buchtitel vom 3. Buch: <strong>{$buch(3)?titel}</strong></p>
```

Infotext: **Bücherliste**

Währung: **EUR**

Bücheranzahl: **3**

Preis vom 1. Buch: **52**

Autor*in vom 2. Buch: **Margit Becher**

Buchtitel vom 3. Buch: **XML – Technologien | Grundlagen | Validierung | Auswertung**

- **Beispiel: Bücherliste (buecher.json)**

- Zusätzliche Datenausgabe als Tabelle mittels xsl:for-each über alle Einträge
- Hinweis: ? = „Lookup-Operator“

```
<xsl:for-each select="1 to $buchanzahl">
  <xsl:variable name="pos" select="."/>
  <tr>
    <td>{$buch?($pos)?nr}</td>
    <td>{$buch?($pos)?autor_in}</td>
    <td>{$buch?($pos)?titel}</td>
    <td>{$buch?($pos)?verlag}</td>
    <td>{$buch?($pos)?jahr}</td>
    <td>{$fn:format-number($buch?($pos)?preis, '#.00')}</td>
    <td>{$buch?($pos)?isbn}</td>
  </tr>
</xsl:for-each>
```

Tabellarische Datenausgabe

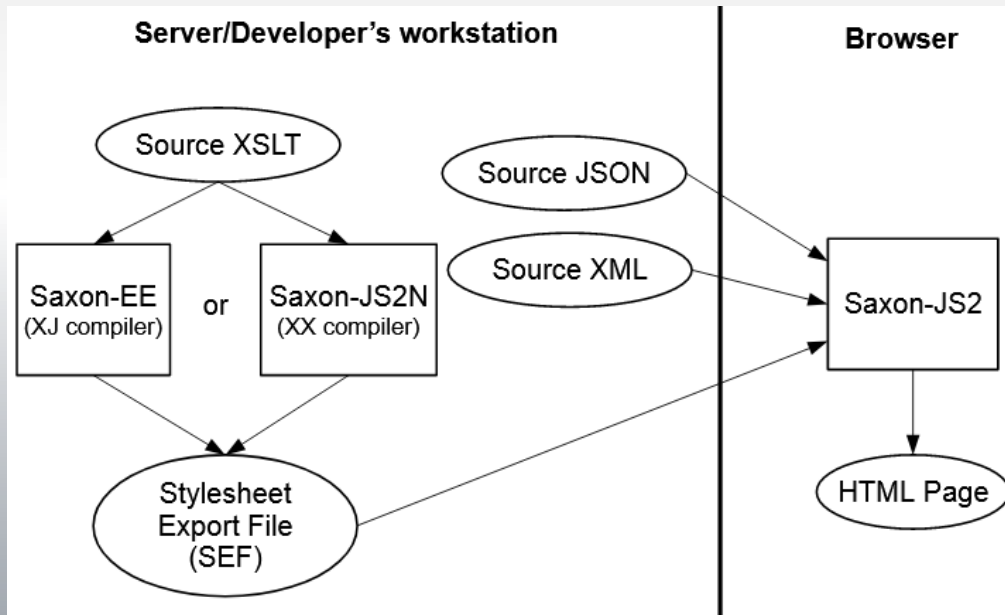
Nr.	Autor*in	Titel	Verlag	Jahr	Preis [EUR]	ISBN
1	Sal Mangano	XSLT Kochbuch	O'REILLY	2006	52.00	978-3-89721-457-6
2	Margit Becher	XML	W3L GmbH	2009	24.90	978-3-937137-69-8
3	Wilfried Grupe	XML – Technologien Grundlagen Validierung Auswertung	mitp-Verlag	2018	29.99	978-3-95845-755-3

(: Saxon-JS :)

Saxon-JS_{1/6}

- Ermöglicht die Nutzung von XSLT 3.0 im Browser und kann XML- sowie JSON-Daten verarbeiten
- Die JS-Dateien bilden die Laufzeitumgebung im Browser (SaxonJS2.rt.js || SaxonJS.min.js)
- XSLT wird vorkompiliert als Stylesheet Export File (SEF) in 1.x als XML und in 2.0 als JSON
- Kommerzieller Saxon-EE nötig (in <oXygen/> enthalten, dort aktuell 9.9.x, erst 10.x kann die JSON-Variante für 2.0 erzeugen)
- Erfordert HTTP-Server (z. B. Apache)
- Frei verfügbar, aber nicht Open Source, insofern an EE-Lizenz gebunden

Prinzip → <https://saxonica.com/>



Saxon-JS 2/6

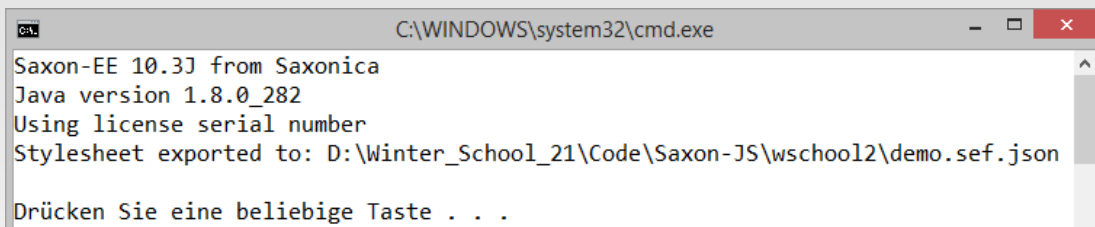
- **Kompilation an der Konsole:**

2.0

```
java -jar X:\Pfad_zu\saxon-ee-10.3.jar -t -xsl:name.xsl -export:name.sef.json -target:JS -nogo  
-relocate:on -ns:##html5
```

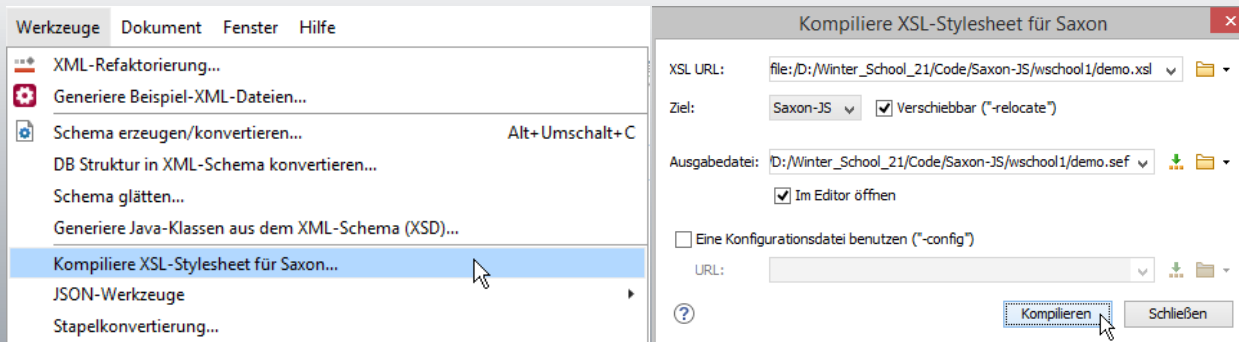
1.2

```
java -jar X:\Pfad_zu\saxon9ee.jar -t -xsl:name.xsl -export:name.sef -target:JS -nogo -relocate:on
```



```
C:\WINDOWS\system32\cmd.exe  
Saxon-EE 10.3J from Saxonica  
Java version 1.8.0_282  
Using license serial number  
Stylesheet exported to: D:\Winter_School_21\Code\Saxon-JS\wschool2\demo.sef.json  
Drücken Sie eine beliebige Taste . . .
```

- **Oder in <oxygen/>
(noch für 1.2):**



Saxon-JS ^{3/6}

- Kompaktdemo:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" lang="de">
  <head>
    <meta charset="UTF-8" />
    <title>Kompaktdemo zu Saxon-JS</title>
    <link rel="stylesheet" href="demo.css" />
    <script src="../../SaxonJS/SaxonJS2.rt.js"></script>
    <script src="demo.js"></script>
  </head>
  <body>
    <header id="header">
      <h1>Kompaktdemo zu Saxon-JS</h1>
    </header>
    <section id="ausgabe1">
      <h2>Ausgabe 1</h2>
    </section>
    <section id="ausgabe2">
      <h2>Ausgabe 2</h2>
    </section>
    <footer id="footer"></footer>
  </body>
</html>
```

index.html

```
window.addEventListener("load", function()
{
  // Initialisierung von Saxon-JS
  var xml_source = "demo.xml";
  var sef_source = "demo.sef.json";

  SaxonJS.transform(
  {
    sourceLocation:      xml_source,
    stylesheetLocation:  sef_source
  }, "async");
});
```

demo.js

```
<?xml version="1.0" encoding="UTF-8"?>
<kategorien>
  <kategorie name="A">
    <inhalt>Inhalt A</inhalt>
  </kategorie>
  <!-- ... -->
  <kategorie name="E">
    <inhalt>Inhalt E</inhalt>
  </kategorie>
</kategorien>
```

demo.xml

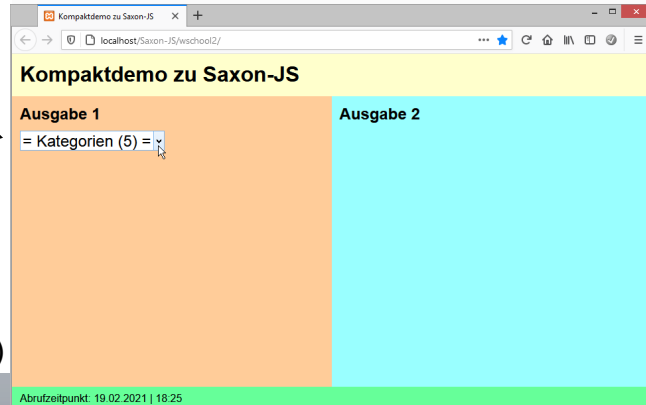
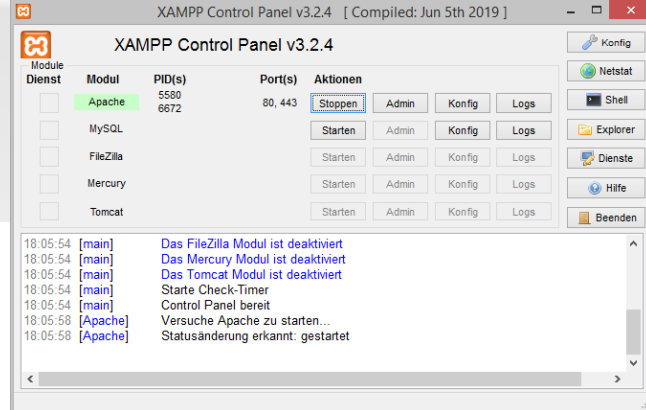
Saxon-JS 4/6

- Kompaktdemo:

```
<!-- Template zur Ausgabe der Kategorien -->
<xsl:template match="kategorien">
  <xsl:result-document href="#ausgabe1" method="ixsl:append-content">
    <form>
      <select id="auswahl">
        <option value="Kategorien">= Kategorien
          ({fn:count(kategorie)}) =</option>
        <xsl:for-each select="kategorie">
          <option value="{@name}">Kategorie {@name}</option>
        </xsl:for-each>
      </select>
    </form>
  </xsl:result-document>

  <xsl:result-document href="#footer" method="ixsl:replace-content">
    <p>Abrufzeitpunkt: {fn:format-dateTime(fn:current-dateTime(),
      '[D01].[M01].[Y0001] | [H01]:[m01]')}</p>
  </xsl:result-document>
</xsl:template>
```

demo.xml (demo.sef.json)



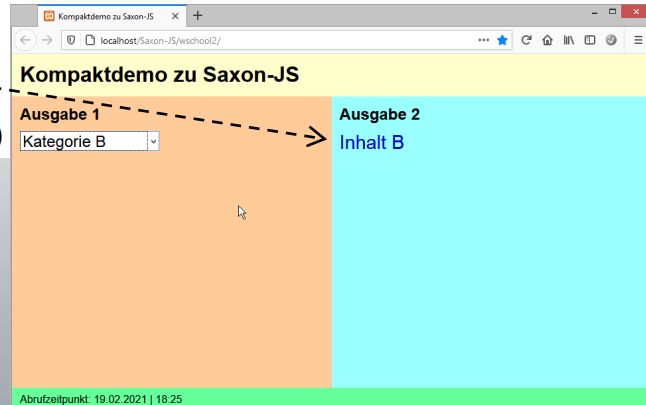
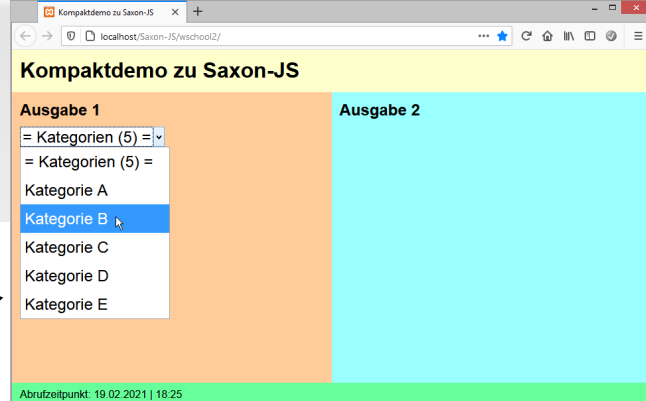
Saxon-JS 5/6

- **Kompaktdemo:**

```
<!-- Template zur Abfrage der Selectbox und Inhaltsausgabe erzeugen -->
<xsl:template match="select[@id eq 'auswahl']" mode="ixsl:onchange">
  <xsl:variable name="curopt" select="." as="element()"/>
  <xsl:variable name="curval" select="ixsl:get($curopt, 'value')"
    as="xs:string"/>

  <xsl:result-document href="#ausgabe2" method="ixsl:replace-content">
    <h2>Ausgabe 2</h2>
    <p>
      {ixsl:source()//kategorie[@name = $curval]/inhalt}
    </p>
  </xsl:result-document>
</xsl:template>
```

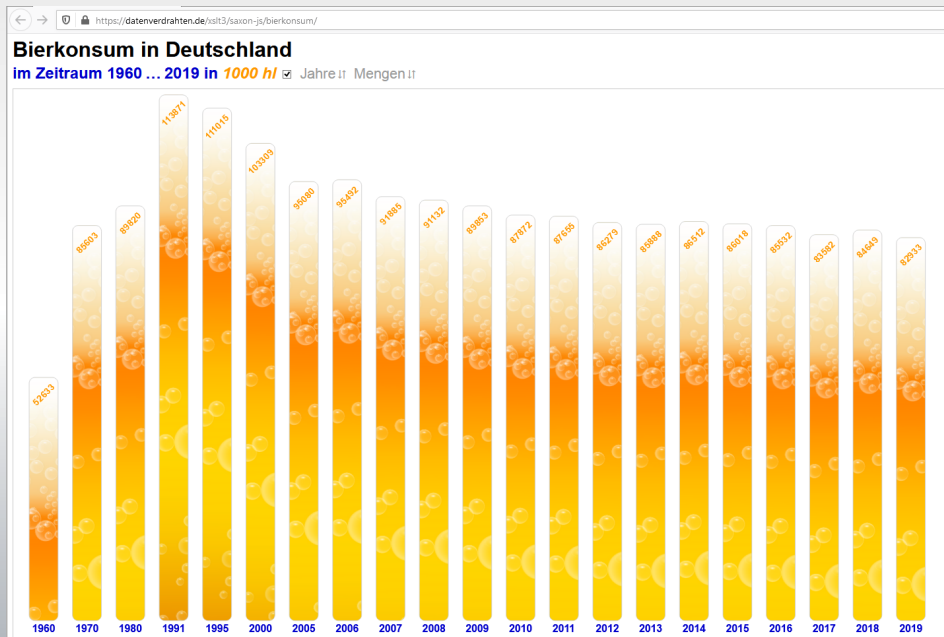
demo.xml (demo.sef.json)



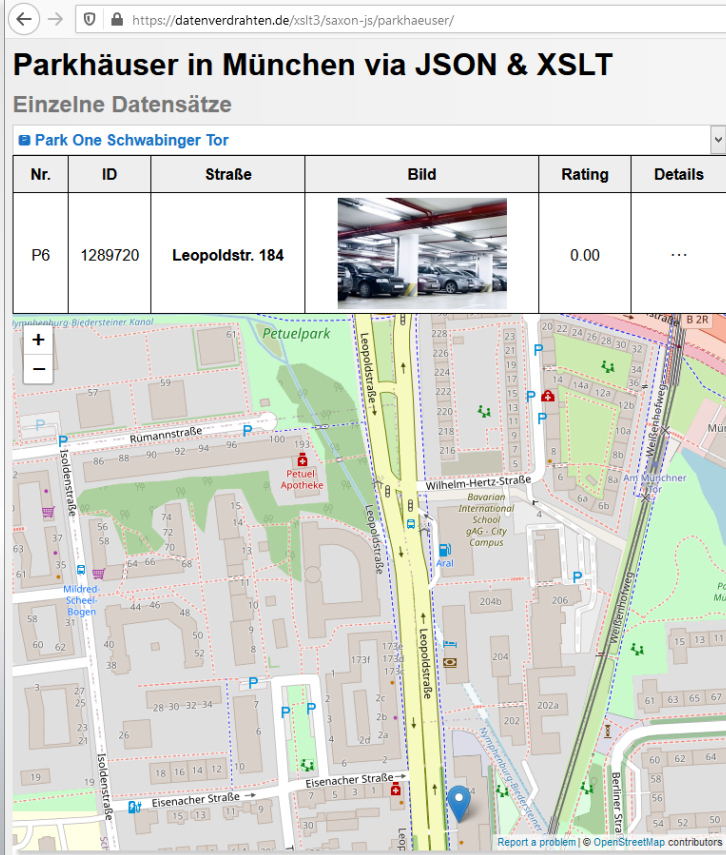
- Ausgaben erfolgen mittels xsl:result-document über IDs
- ixsl:Erweiterungen triggern Events (IXSL = Interactive XSL)

Saxon-JS_{6/6}

- Online-Beispiele:



<https://datenverdrahten.de/xslt3/saxon-js/bierkonsum/>



<https://datenverdrahten.de/xslt3/saxon-js/parkhaeuser/>

(: Zugabe :)

Zugabe

- XPath 3.1 ermöglicht Ermittlung von Zufallswerten, analog zu Math.random() in JavaScript
 - fn:random-number-generator() $\rightarrow 0 \leq \text{Wert} < 1$
 - **Beispiel: Zufallsfarben in rgb()-Schreibweise**

```
<xsl:variable name="myrng" select="fn:random-number-generator()" as="map(xs:string, item())"/>

<xsl:variable name="rgb" as="xs:integer*">
  <xsl:value-of select="xs:integer(fn:floor($myrng?number * 256))"/>
  <xsl:value-of select="xs:integer(fn:floor($myrng?next()?number * 256))"/>
  <xsl:value-of select="xs:integer(fn:floor($myrng?next()?next()?number * 256))"/>
</xsl:variable>

<p>R: <xsl:value-of select="$rgb[1]"/></p>
<p>G: <xsl:value-of select="$rgb[2]"/></p>
<p>B: <xsl:value-of select="$rgb[3]"/></p>
<p><strong>{"rgb(" || $rgb[1] || ',' || $rgb[2] || ',' || $rgb[3] || ") "}</strong>
  <span style="background-color: rgb({$rgb[1]},{$rgb[2]},{$rgb[3]})">&#xA0;</span>
</p>
```

RGB-Zufallsfarben

R: 61

G: 190

B: 25

rgb(61,190,25)



Danke für Ihr Interesse!