

DVD auf
S.3



POWER-DVD

**20 STUNDEN
WEBWISSEN**

S&S

entwickler
magazin

Deutschland €6,50 Österreich €7,00 Schweiz sFr13,40

entwickler

www.entwickler-magazin.de

magazin

November/Dezember 6.2013

ANSI C

XSLT

Tuning für
den Browser

GWT

RWD mit GWT

**Requirements
Engineering**

Mehr Professionalität
in Scrum

**Objektorientiertes
Programmieren**

**Testen von
Embedded Systems**

3-D-Druck

**Konstruktion, Slicen
und Vorbereitung**

OpenGL

**Geburt eines
Sonnensystems**

Datenträger enthält
Info- und
Lehrprogramme
gemäß § 14 JuSchG



4 194156 606509

0 6



©istockphoto.com/teekid

Einstieg in die Webentwicklung mit Saxon-CE und XSLT 2.0

XSLTuning für Browser

Der Einsatz von XSLT dürfte für viele Webentwickler entweder ein alter Hut oder weitgehend unbekannt sein. Dennoch erweist sich diese Technologie durch die Verfügbarkeit von Saxon-CE als Bereicherung des Methodenspektrums. Im Folgenden werden die wesentlichen Voraussetzungen und Konzepte vorgestellt und mit einer Beispielanwendung demonstriert.

von Dr. Thomas Meinike

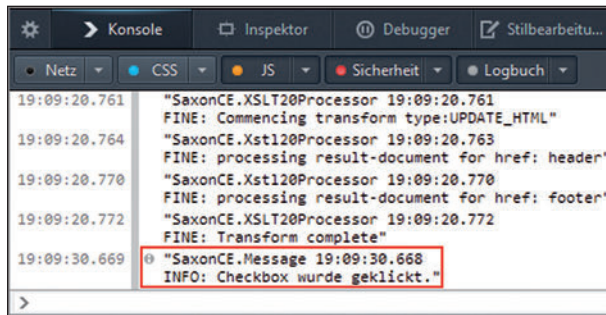
XSLT 1.0 wurde bereits 1999 vom W3C spezifiziert [1] und ist heute überwiegend im Bereich der Technischen Dokumentation gebräuchlich. In Redaktions- und Content-Management-Systemen sowie bei Dokumentationsprozessen wird diese Technologie zur Erstellung von Medienprodukten ausgehend von XML-Daten eingesetzt. Dazu gehören beispielsweise E-Books im EPUB-Format [2]. Webbrowser unterstützen die Nutzung von XSLT beim Laden von XML-Dokumenten mit eingebundener Verarbeitungsanweisung `<?xml-stylesheet href="name.xml" type="text/xsl"?>`. Alternativ lassen sich XML- und XSLT-Dokumente mittels JavaScript nachladen und direkt transformieren, wobei die konkreten Implementierungen browserspezifisch sind. Auch auf der Serverseite wird XSLT durchaus produktiv eingesetzt, u. a. als PHP-Modul [3]. Zum Einstieg in das Thema XSLT und XPath ist noch immer das SELF-HTML-Tutorial [4] zu empfehlen.

Seit 2007 steht XSLT 2.0 mit wesentlichen Erweiterungen der Programmiermöglichkeiten zur Verfügung. Dazu gehören Techniken zur Gruppierung, umfangreiche Datentypen, Einsatz eigener Funktionen, und es sind vielfältige XPath-Funktionen hinzugekommen. Einen kompakten Überblick hat der Autor im Rahmen eines Vortrags gegeben [5]. Browserhersteller haben jedoch bisher keine Anstrengungen unternommen, die 2.0-Möglichkeiten verfügbar zu machen. Im Gegenteil, kürzlich haben die Entwickler der Chromium-Engine Blink durchblicken lassen, vom XSLT-Support ganz abzusehen [6].

Retter in der Not?

Bereits 2011 hat Michael Kay – bekannt als Herausgeber beim W3C, Buchautor und Entwickler des populären Saxon-Prozessors – mit seinem Saxonica-Team die erste Version der Software Saxon-CE vorgestellt [7]. Diese Fassung wurde kommerziell vertrieben und war an einzelne Domains gebunden, was offenbar keinen größeren

Abb. 1:
Firefox-
Konsole
(CTRL +
SHIFT + I)



Erfolg nach sich zog. Im Februar 2013 wurde die Version 1.1 unter einer Open-Source-Lizenz (MPL) veröffentlicht. Das Projekt ist ebenfalls bei GitHub präsent [8].

Saxon-CE bildet die XSLT-Fähigkeiten des in Java entwickelten Prozessors Saxon als JavaScript-Bibliothek ab. Die konkrete Cross-Compiler-Umsetzung erfolgte mit dem Google Web Toolkit (GWT) und diversen Anpassungen. Details dazu liefert [9].

Zutaten

Das von GitHub geladene Archiv *Saxon-CE-master.zip* enthält die Software selbst sowie Beispielcode und die Entwicklerdokumentation. Letztere steht auch online zur Verfügung und wurde ebenfalls mit Saxon-CE aufbereitet [10].

Wesentlich für den Produktiveinsatz ist die Einbindung der im Verzeichnis *Saxonce* enthaltenen JavaScript-Ressource *Saxonce.nocache.js* im *head*-Element eines HTML-Dokuments. Dieses initiale Skript lädt browserspezifische **.cache.html*-Dateien mit dem eigentlichen CE-Code nach. Deren Größe von ~880 KB

mag zwar etwas erschrecken, aber nach dem ersten Laden greift das lokale Caching im Browser. Entwickler sollten sich dem Inhalt des analog aufgebauten Verzeichnisses *SaxonceDebug* zuwenden und

erhalten über die üblichen Browserkonsolen wertvolle Hinweise und erweiterte Fehlermeldungen (Abb. 1). Die Einbindung der JavaScript-Ressource *Saxonce.nocache.js* mit Zuweisung der zu ladenden XML- und XSLT-Dokumente in ein gewöhnliches HTML-Dokument veranschaulicht Listing 1. Weitere sinnvolle Komponenten wie CSS-Referenzen wurden hier ausgespart.

Das im Code ersichtliche *logLevel* kann mit diesen Werten von „zurückhaltend“ bis „geschwätzig“ eingestellt werden: *SEVERE*, *WARNING*, *INFO*, *FINE*, *FINER*, *FINEST*. Die dargestellte Einbindungsform erlaubt die Anreicherung mit weiteren Methoden, und es lassen sich bei Bedarf auch mehrere Transformationen schrittweise ausführen. Sofern im Regelfall nur XML- und XSLT-Dokumente übergeben werden sollen, kann das zweite *script*-Element kompakter formuliert werden:

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

In der Dokumentation ist ein zusätzliches Attribut *language="xslt2.0"* angegeben, das bei einer HTML-Validierung jedoch als ungültig moniert wird. Offenbar hat das Weglassen keine Auswirkungen auf die Funktionalität der Programmlogik. Der nun fehlende *logLevel*-Parameter kann in Form von *?logLevel=INFO* einfach an den URL angehängt werden. Angemerkt sei, dass die Verwendung einer XML-Datenstruktur optional ist, für alternative Anwendungen also lediglich eine passende Ausgaben erzeugende Transformation ausreicht. Falls mit der Ausführung eines bestimmten benannten Templates begonnen werden soll, bietet sich die Angabe von *data-initial-template="..."* an. Man kann als Spezialfall auch die eingangs genannte XSLT-Einbindung via *<?xml-stylesheet ...?>* praktizieren und gewissermaßen ein Boot-Style-Sheet mit 1.0-Code laden, das ein HTML-Dokument in Form von Listing 1 erzeugt und schließlich das 2.0-Style-Sheet nachlädt.

Listing 1

```
<!DOCTYPE html>
<html lang="de">
  <head>
    <meta charset="UTF-8" />
    <title>CE-Basis</title>
    <script type="text/javascript" src=
      "../Saxonce/Saxonce.nocache.js"></script>
    <script type="text/javascript">
      var onSaxonLoad = function() {
        Saxon.run(
        {
          stylesheet: "name.xml",
          source: "name.xml",
          logLevel: "INFO"
        });
      };
    </script>
  </head>
  <body>
    <!-- weitere Inhalte ... -->
  </body>
</html>
```

Browser	Einstellungen
Chrome/Iron	Programm mit der Option <code>--allow-file-access-from-files</code> starten.
Internet Explorer	Nach dem Laden des HTML-Dokuments im IE10 ggf. „Gebrochene Inhalte zulassen“ bestätigen. Mit F12 die Entwicklertools aufrufen und „Browsermodus: IE10“ belassen, jedoch bei „Dokumentmodus:“ von „Standards“ auf „IE8-Standards“ umstellen (kurz ALT + 8). Bis zum nächsten Browserstart wird die lokale Transformation ausgeführt. Alternative: Im <i>head</i> -Element dieses <i>meta</i> -Element platzieren: <pre><meta http-equiv="X-UA-Compatible" content="IE=EmulateIE8"/></pre> Zusätzlich unter INTERNETOPTIONEN ERWEITERT SICHERHEIT diese Option aktivieren: „Ausführung aktiver Inhalte in Dateien auf dem lokalen Computer zulassen“.
Firefox	<i>about:config</i> aufrufen und bei der Option <i>security.fileuri.strict_origin_policy</i> den Wert von <i>true</i> auf <i>false</i> setzen.
Opera	<i>opera:config</i> aufrufen und unter <i>UserPrefs</i> die Option <i>Allow File XMLHttpRequest</i> anhaken und speichern.
Safari	Keine Änderungen nötig.

Tabelle 1: Browserkonfiguration auf Entwicklungsrechnern

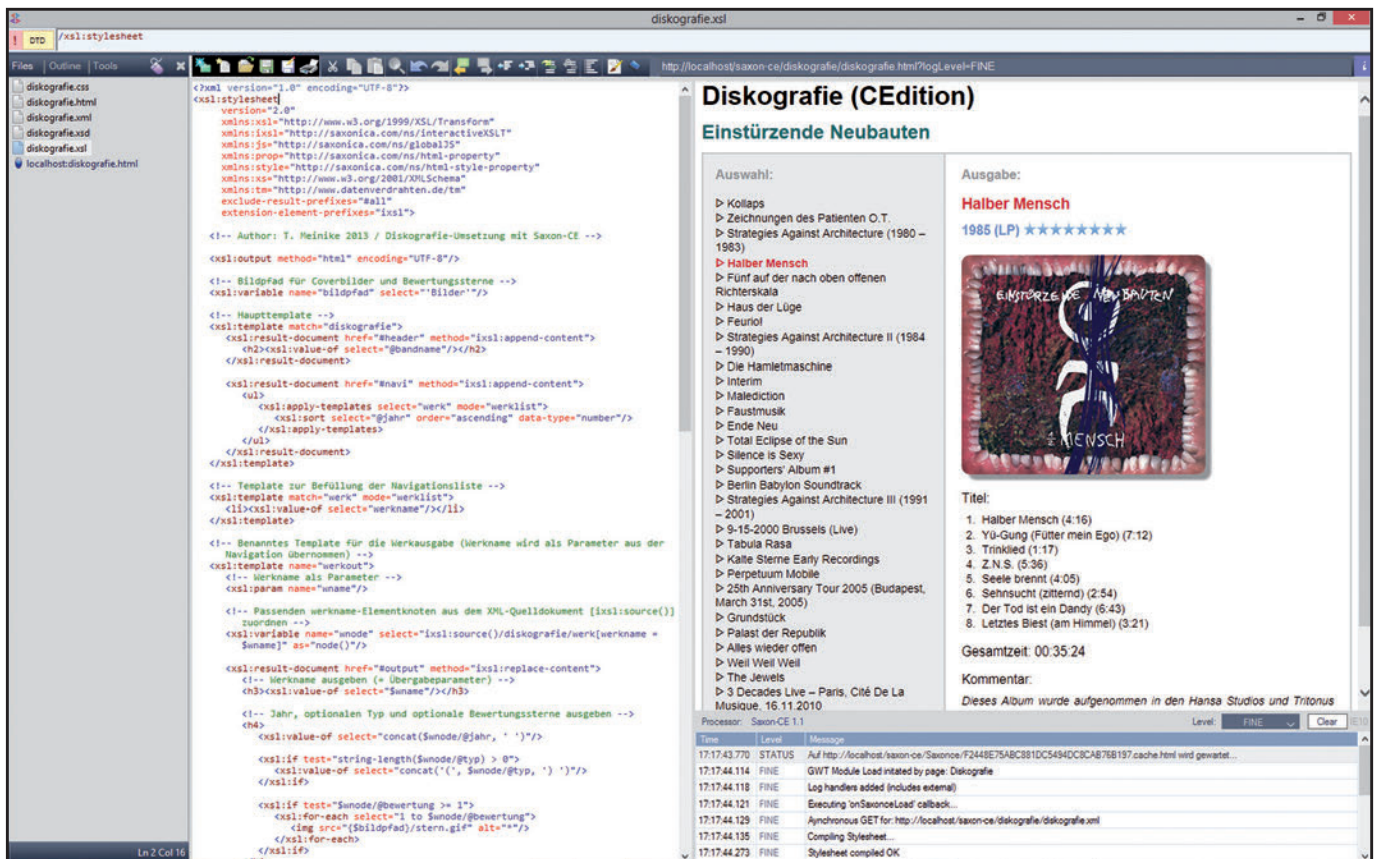


Abb. 2: XMLQuire im Einsatz

Für die Entwicklung selbst genügen die üblichen HTML- bzw. XML-Editoren völlig. Der Aufruf der Seiten erfolgt mit konventionellen Webbrowsern, wobei auf den Entwicklungsrechnern gegebenenfalls die in Tabelle 1 genannten, experimentell ermittelten Konfigurationen vorzunehmen sind.

Etwas komfortabler gestaltet sich die Arbeit unter Windows mit dem Werkzeug XMLQuire [11], das Code, Browservorschau und Log-Bereich unter einer Oberfläche verfügbar macht. Hinweis: Da das Programm standardmäßig den IE7-Modus einschaltet, sollte es einmalig mit Administratorrechten ausgeführt werden. Danach wird auch im Benutzermodus der aktuelle System-IE verwendet. Allerdings wird ein lokaler Webserver vorausgesetzt, etwa auf der Basis des bekannten XAMPP-Pakets [12]. **Abbildung 2** zeigt XMLQuire mit einem Beispiel des Autors, das als Erweiterung eines umfangreichen XML-basierten Banddiskografie-Projekts für Lehrzwecke mit vielfältigen Ausgabemöglichkeiten bis hin zu E-Book-, InDesign- und Word-Formaten entwickelt wurde. Weitere Informationen sind unter [13] verfügbar.

Programmlogik

Das HTML-Grundgerüst enthält noch keinen Hinweis auf die eigentlichen Inhalte im *body*-Element und wie diese aus dem XSLT-Kontext dorthin gelangen. Listing 2 zeigt einige verschachtelte *div*-Elemente, die nachfolgend mit Inhalten befüllt werden. Dieser Ansatz wurde im unter [14] abrufbaren Projekt zur Darstel-

lung statistischer Daten in HTML-Tabellenform und als SVG-Kreisdiagramm verwendet (siehe **Abb. 3** und die Hinweise im Kasten „Zum Beispielprojekt“).

Der Schlüssel zur Kommunikation mit dem XSLT-Code liegt in den vergebenen IDs. Diese werden in Form von *#idname* angesprochen. XSLT 2.0 stellt das Element *xsl:result-document* zur Verfügung, das üblicherweise für mehrfache Dateiausgaben innerhalb einer Transformation bestimmt ist. Saxon-CE erweitert seine Funktion, um gezielt Dokumentfragmente in das HTML-DOM einzubringen. Das *div*-Element mit *id="footer"* wird so angesprochen:

```
<xsl:result-document href="#footer" method="ixsl:append-content">
  <p>[ Projekt by ... ]</p>
</xsl:result-document>
```

Der Wert des *method*-Attributs gibt an, ob der Inhalt an den vorhandenen angehängt oder dieser komplett überschrieben werden soll (Wert *ixsl:replace-content*). Das Präfix *ixsl* verweist auf eine XSLT-Erweiterung für Interaktionen, die mit dem Namensraum *xmlns:ixsl="http://saxonica.com/ns/interactiveXSLT"* verknüpft ist. Dem dargestellten Footer-Bereich wird somit ein neuer Absatz hinzugefügt.

Listing 2

```
<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>
```

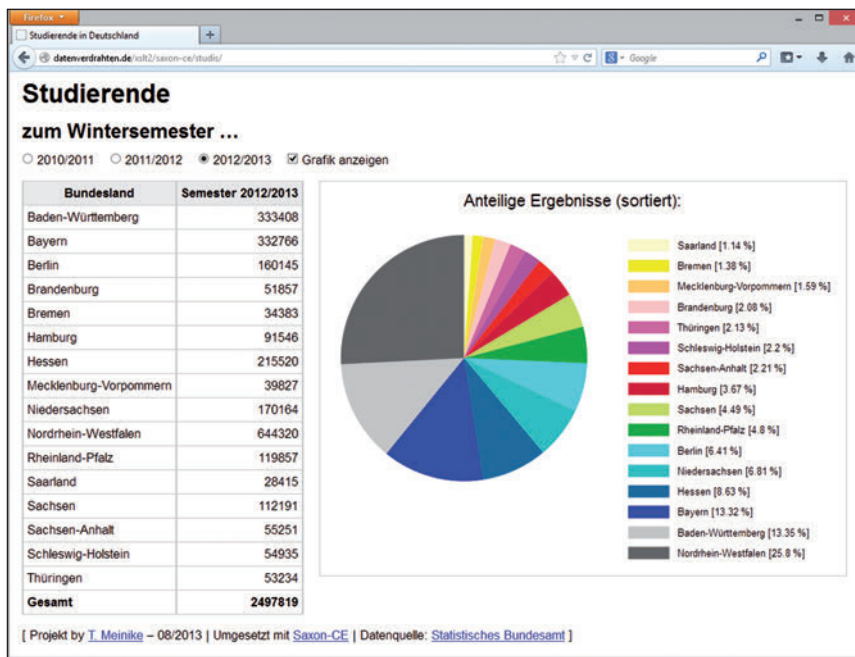


Abb. 3: Mit Saxon-CE umgesetztes Beispielprojekt

Den Rahmen des XSLT-Style-Sheets bildet wie üblich ein `xsl:stylesheet`-Element mit Versionsnummer und Namensraumdeklarationen sowie den Templates als funktionale Einheiten:

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

In den Templates wird auf weiterführende Templates mit den Konstrukten `xsl:apply-templates` oder `xsl:call-template` verwiesen, also eine XSLT-Entwicklern vertraute Vorgehensweise. Die genannten `xsl:result-document`-Blöcke werden an den jeweils passenden Stellen für die gewünschten Ausgaben eingebunden.

Listing 3

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

Zum Beispielprojekt

Der vollständige Code des Beispielprojekts [14] lässt sich am einfachsten direkt im Browser einsehen. Beim Aufruf der Quelltextansicht werden die Links zum XML-Dokument mit den verwendeten Daten und zum ca. 200 Zeilen umfassenden XSLT-Style-Sheet sichtbar (*studierende.xml* bzw. *studierende.xsl*).

Es empfiehlt sich insbesondere die Beschäftigung mit der Template-Funktionalität zur Formularverarbeitung und zur in den HTML-Code integrierten SVG-Ausgabe sowie für die Sortierung der Tabelle beim Anklicken ihrer Spaltenüberschriften.

Benutzeraktionen

Die Reaktion auf typische Benutzeraktionen wie das Anklicken eines Objekts wird ebenfalls über die *ixsl*-Erweiterung direkt im Style Sheet realisiert. Dazu wird dem jeweiligen Template ein speziell formuliertes *mode*-Attribut zugewiesen, etwa *mode="ixsl:onclick"*. Es ist leicht nachzuvollziehen, dass *onclick* hier einen aus JavaScript bekannten Event Handler darstellt und sich nach Bedarf analog weitere wie *onchange*, *onkeydown*, *onmouseover* usw. formulieren lassen. Der Code von Listing 3 ermöglicht das Ausblenden des *div*-Elements mit *id="grafik"* über eine Formular-Checkbox.

Dabei ist ein weiteres Namensraumpräfix involviert: *prop* erlaubt die Abfrage von DOM-Eigenschaften (@ fragt in XPath-Ausdrücken Attribute ab), während *ixsl:set-property* als Erweiterungselement fungiert. In purem JavaScript steht dieses Codefragment für:

```
document.getElementById('grafik').style.visibility = 'hidden';
```

Spezielle Techniken

Das CE-Konzept zielt nicht nur auf die Programmierung mit XSLT ab, sondern ermöglicht auch den Zugriff auf die JavaScript-Schnittstelle des Browsers. Dazu dienen Erweiterungen wie *ixsl:call()*, *ixsl:eval()*, *ixsl:event()*, *ixsl:get()* und *ixsl:window()*. Mit diesen Funktionen lassen sich Objekte und ihre Methoden ansprechen sowie ausgelöste Events abfangen und verarbeiten. Ein kleines Beispiel: Die Berechnung des Sinuswerts der Zahl 1 erfolgt mit *Math.sin(1)* unter JavaScript. Sinus- und Cosinus-Berechnungen spielen beim Aufbau der Pfade im SVG-Kreisdiagramm eine tragende Rolle. XSLT/XPath 2.0 unterstützt noch keine Winkelfunktionen (jedoch die in der Endphase der Spezifizierung befindliche Version 3.0). Ersatzweise bietet sich der Zugriff auf die nativen Browserfunktionen an. Mit den genannten Erweiterungen kommt man so an den gesuchten Sinuswert:

```
ixsl:call(ixsl:get(ixsl:window()), 'Math'), 'sin', 1)
```

Für die mehrfache Anwendung ist diese Form etwas sperrig. Alternativ ließe sich daraus mittels *xsl:function* eine benutzerdefinierte Funktion erstellen. Im Beispielprojekt wurde ein anderer Weg gewählt: Über den *js*-Namensraum (*xmlns:js="http://saxonica.com/ns/globalJS"*) lassen sich im XSLT auch JavaScript-Funktionen aus dem HTML-Kontext aufrufen. Dazu dient eine separate JavaScript-Datei (*zusatz.js*) mit diesem Inhalt:

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

Der Aufruf kann nun in der einfach handhabbaren Form *js:sin(1)* erfolgen. Obwohl für die grundsätzliche Arbeit keine JavaScript-Kenntnisse nötig sind, erweisen sich diese als vorteilhaft, um alle Möglichkeiten der clientseitigen Webentwicklung auszureizen.

Beim Studieren der Dokumentation finden sich weitere hilfreiche Konstrukte wie *ixsl:page()* und *ixsl:source()* zum parallelen Zugriff auf den erzeugten HTML-Baum und auf das zugrunde liegende XML-Quelldokument. Beim genannten Diskografieprojekt wird *ixsl:page()* für die Abfrage des jeweils aktivierten Eintrags der zuvor aus dem XML-Inhalt erzeugten dynamischen Navigationsliste eingesetzt (siehe den Auswahlbereich in **Abb. 2**). Falls zeitlich verzögerte Templateausführungen benötigt werden, kann auf das Element *ixsl:schedule-action* mit einem *wait*-Attribut zur Angabe der Verzögerung in Millisekunden zurückgegriffen werden.

Fazit und Ausblick

Saxon-CE erfindet die Webentwicklung nicht neu, geht aber einen sehr interessanten Weg. Natürlich darf man keine Vorbehalte gegen die verwendeten XML-Technologien haben, um sich diese Entwicklungsmethode zu erschließen. Im Verlauf der Recherchen zu diesem Artikel wurde das kreative Potenzial der genannten Techniken immer deutlicher. Wohin die Reise letztlich geht, bleibt abzuwarten. Zur Umsetzung von Onlinehilfesystemen oder sonstigen Seiten mit Schwerpunkt auf Darstellung und Dokumentation von Daten ist Saxon-CE auf jeden Fall gut gerüstet. Dem Blogbeitrag unter [15] nach zu urteilen, ist das Interesse an einem clientseitigen XSLT-Revival nicht nur beim Autor geweckt. Unter diesen Vorzeichen kann man vermeintlichen Plänen zur Abschaltung von XSLT-1.0-Browser-Engines gelassen entgegensetzen.

Anzeige



Dr. Thomas Meinike ist seit 1997 an der Hochschule Merseburg als Lehrkraft tätig. Seine Arbeitsschwerpunkte sind XML-Anwendungen in der Technischen Dokumentation, Onlinehilfen und Webentwicklung.



thomas.meinike@hs-merseburg.de

Links & Literatur

- [1] W3C: XSL Transformations (XSLT): <http://www.w3.org/TR/xslt>
- [2] Meinike, T.: E-Book-Projekt epubMinFlow: <http://datenverdrahten.de/epubMinFlow/>
- [3] PHP-Manual: <http://php.net/manual/de/ref.xslt.php>
- [4] SELFHTML: <http://de.selfhtml.org/xml/darstellung/>
- [5] Meinike, T.: Syntax-Highlights von XSLT 2.0 und XPath 2.0, AJAX IN ACTION 2007: http://www.iks.hs-merseburg.de/~meinike/PDF/AIA2007_XSLT_XPath_20_Meinike.pdf
- [6] blink-dev: Intent to Deprecate and Remove: XSLT: <http://goo.gl/1VmqH4>
- [7] Saxon-CE: <http://saxonica.com/ce/index.xml>
- [8] GitHub: <https://github.com/Saxonica/Saxon-CE>
- [9] Delpratt, O. and Kay, M.: Interactive XSLT in the browser – Balisage: The Markup Conference 2013: <http://balisage.net/Proceedings/vol10/html/Delpratt01/BalisageVol10-Delpratt01.html>
- [10] CE-Dokumentation: <http://saxonica.com/ce/user-doc/1.1/>
- [11] XMLQuire Web Edition – A Free XSLT 2.0 Editor for the Web: <http://qutoric.com/xmlquire/ce/>
- [12] XAMPP: <http://www.apachefriends.org/>
- [13] Meinike, T.: Material zum Projekt Banddiskografie: <http://www.iks.hs-merseburg.de/~meinike/vortraege.php>
- [14] Meinike, T.: Studierende in Deutschland: <http://datenverdrahten.de/xslt2/saxon-ce/studis/>
- [15] O'Reilly Media: <http://programming.oreilly.com/2013/08/using-xslt-2-0-as-a-web-scripting-language.html>