

Experimentelles zu XSLT und XPath 4.0

Dr. Thomas Meinike

Hochschule Merseburg
Fachbereich Wirtschaftswissenschaften
und Informationswissenschaften



09.11.2022

Einstieg ^{1/4}

- **XSLT = Extensible Stylesheet Language Transformations** zur Verarbeitung von XML und weiteren Ausgangsformaten (u. a. JSON).
 - Versionen:
 - ✓ 1999: 1.0
 - ✓ 2007: 2.0
 - ✓ 2017: 3.0
 - ✗ 202?: 4.0
 - **XPath = Extensible Path Language** zum Zugriff auf die Dokumentstrukturen im Kontext von XSLT und XQuery.
 - Versionen:
 - ✓ 1999: 1.0
 - ✓ 2007: 2.0
 - ✓ 2017: 3.1
 - ✗ 202?: 4.0
- Die Spezifikationen der 4.0-Versionen befinden sich noch im Aufbau.
- Erste Ansätze lassen sich bereits mit dem XSLT-Prozessor Saxon-EE (aktuell 11.4) untersuchen.
- Ergebnisse und Erfahrungen mit voraussichtlich verfügbaren Techniken werden vorgestellt.

Einstieg ^{2/4}

- Michael Kay (Saxonica) präsentierte auf der Konferenz XML Prague 2020 das Thema »A Proposal for XSLT 4.0«, welches die Erweiterung der Standards auf den Weg brachte.
<https://www.saxonica.com/papers/xmlprague-2020mhk.pdf>

- W3C setzte XSLT Extensions Community Group ein.
<https://www.w3.org/community/xslt-40/>

Group's public email, repo and wiki activity over time



- Erster Draft: <https://www.saxonica.com/qt4specs/XT/Overview-diff.html>

XSL Transformations (XSLT) Version 4.0

W3C Editor's Draft 11 November 2020



- Aktuelle Entwicklungen unter <https://qt4cg.org/> verfolgbar.

Specifications

The latest drafts of

- [XSLT Transformations \(XSLT\) Version 4.0 \(latest diffs\)](#)
- [XPath and XQuery Functions and Operators 4.0 \(latest diffs\)](#)
- [XML Path Language \(XPath\) 4.0 \(latest diffs\)](#), and
- [XQuery 4.0: An XML Query Language \(latest diffs\)](#)
- [XQuery and XPath Data Model 4.0 \(latest diffs\)](#)
- [XSLT and XQuery Serialization 4.0 \(latest diffs\)](#)

XSL Transformations (XSLT) Version 4.0

W3C Editor's Draft 31 October 2022



XPath and XQuery Functions and Operators 4.0

W3C Editor's Draft 31 October 2022



XML Path Language (XPath) 4.0 WG Review Draft



Stand: 04.11.2022 W3C Editor's Draft 31 October 2022

Einstieg 4/4

- Basis der durchgeführten Untersuchungen:

→ Saxon-EE 11.4 (kommerziell) + Dokumentation
<https://saxonica.com/>

→ Erweiterung für Visual Studio Code von DeltaXML
<https://deltaxml.github.io/vscode-xslt-xpath/>



XSLT/XPath for Visual Studio Code v1.3.7

DeltaXML | 32.497 | ★★★★★ (6)

Comprehensive language support for XSLT 3.0 / XPath 3.1

Deaktivieren

Deinstallieren

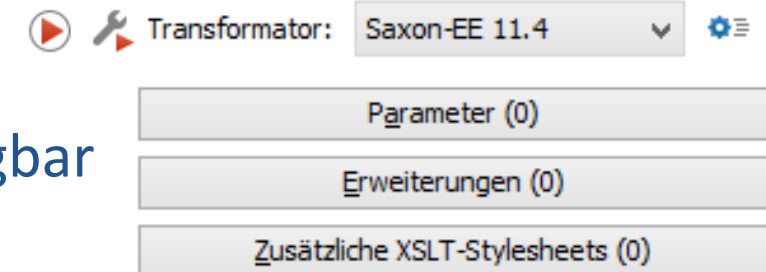


Experimental 4.0 extensions

- ▶ New types
- ▶ New functions
- ▶ New function syntax
- ▶ Other XPath 4.0 extensions
- ▶ XSLT 4.0 extensions

Saxon-Pfad: workspace.code-workspace
XML, XSLT, Optionen: .vscode/tasks.json
Build/Transformation: Strg+Umschalt+B

→ Seit Version 25.0 (10/22) auch im <oXygen/> XML Editor verfügbar
<https://www.oxygenxml.com/>



Ausgewählte Techniken ^{1/26}

- Neue Typen – Enum (1)

✓ Ermöglicht Abfragen über Aufzählungen

```
<xsl:variable name="tech" select="'SVG'" as="xs:string"/>

<p>Technologie-Skills von {$tech} sind
  <xsl:choose>
    <xsl:when test="$tech instance of enum('CSS', 'HTML', 'JS', 'MathML', 'SVG', 'XML', 'XPath', 'XSLT')"
      select="'vorhanden'"/>
    <xsl:otherwise select="'nicht vorhanden'"/>
  </xsl:choose>.
</p> <!-- vorhanden -->
```

Enum Type

Technologie-Skills von SVG sind vorhanden.

Ausgewählte Techniken 2/26

- Neue Typen – Enum (2)

✓ Ermöglicht Abfragen über Aufzählungen

```
<xsl:function name="tm:enum_test" as="xs:string">
  <xsl:param name="s" as="xs:string"/>
  <xsl:if test="$s instance of enum('A', 'B', 'C', 'D')" then="'ja'" else="'nein'"/>
</xsl:function>
```

```
<p>Kommt C vor? {tm:enum_test("C")}</p> <!-- ja -->
<p>Kommt X vor? {tm:enum_test("X")}</p> <!-- nein -->
```

Enum Type

Kommt C vor? ja

Kommt X vor? nein

Ausgewählte Techniken ^{3/26}

- Neue Typen – Union

✓ Kombinierte Abfrage von Datentypen (hier xs:date und xs:dateTime)

```
<!-- Rückgabe des Datums für beide Eingabe-Typen -->
<xsl:function name="tm:union_test" as="xs:string">
  <xsl:param name="union_arg" as="union(xs:date, xs:dateTime)"/>
  <xsl:sequence select="fn:substring(xs:string($union_arg), 1, 10)"/>
</xsl:function>
```

```
<xsl:variable name="d" select="fn:current-date()"/>
<xsl:variable name="dt" select="fn:current-dateTime()"/>
```

```
<p>{tm:union_test($d)}</p> <!-- YYYY-MM-DD -->
<p>{tm:union_test($dt)}</p> <!-- YYYY-MM-DD -->
```

Union Type

2022-11-09

2022-11-09

Ausgewählte Techniken 4/26

- Neue Typen – Record (1)
 - ✓ Deklaration von Datentypen als listenartige Objekte / Mengen mittels Maps (ursprünglich Tuple genannt, u. a. für Koordinaten, Vektoren geeignet)
 - ✓ Michael Kay gibt in der Saxon-Doku ein Beispiel für komplexe Zahlen mit Record Type `cx:complex()` und `cx:add()` zur komplexen Addition an
 - ✓ Funktionen `cx:sub()`, `cx:mul()`, `cx:div()` wurden nachgebaut / ergänzt

```
<!-- Komplexer Zahlentyp cx:complex() mit Real- und Imaginär-Teil | Zugriff mit $var?r und $var?i -->
<xsl:function name="cx:complex" as="record(r as xs:double, i as xs:double)">
  <xsl:param name="real" as="xs:double"/>
  <xsl:param name="imag" as="xs:double"/>
  <xsl:sequence select="map{'r':$real, 'i':$imag}"/>
</xsl:function>
```

- Neue Typen – Record (2)

- ✓ Komplexe Zahlen

```
<!-- Funktion cx:add() -->
<xsl:function name="cx:add" as="record(r as xs:double, i as xs:double)">
  <xsl:param name="x" as="record(r as xs:double, i as xs:double)"/>
  <xsl:param name="y" as="record(r as xs:double, i as xs:double)"/>
  <xsl:sequence select="cx:complex($x?r + $y?r, $x?i + $y?i)"/>
</xsl:function>
```

```
<!-- Funktionen cx:sub(), cx:mul(), cx:div() sind analog aufgebaut, hier nur die Rechenschritte -->
<xsl:sequence select="cx:complex($x?r - $y?r, $x?i - $y?i)"/>
```

```
<xsl:sequence select="cx:complex($x?r * $y?r - $x?i * $y?i, $x?r * $y?i + $x?i * $y?r)"/>
```

```
<xsl:sequence select="cx:complex(($x?r * $y?r + $x?i * $y?i) div ($y?r * $y?r + $y?i * $y?i),
                                ($x?i * $y?r - $x?r * $y?i) div ($y?r * $y?r + $y?i * $y?i))"/>
```

- Neue Typen – Record (3)

- ✓ Komplexe Zahlen

```
<!-- Rechendemo -->
```

```
<xsl:variable name="czahl1" select="cx:complex(4, 6)"/>
```

```
<xsl:variable name="czahl2" select="cx:complex(8, -3)"/>
```

```
<xsl:variable name="adderg" as="map(*)" select="cx:add($czahl1, $czahl2)"/>
```

```
<xsl:variable name="suberg" as="map(*)" select="cx:sub($czahl1, $czahl2)"/>
```

```
<xsl:variable name="mulerg" as="map(*)" select="cx:mul($czahl1, $czahl2)"/>
```

```
<xsl:variable name="diverg" as="map(*)" select="cx:div($czahl1, $czahl2)"/>
```

```
<p>(4 + 6i) + (8 + -3i) = {$adderg?r} + {$adderg?i}i</p>
```

```
<p>(4 + 6i) - (8 + -3i) = {$suberg?r} + {$suberg?i}i</p>
```

```
<p>(4 + 6i) * (8 + -3i) = {$mulerg?r} + {$mulerg?i}i</p>
```

```
<p>(4 + 6i) / (8 + -3i) = {$diverg?r} + {$diverg?i}i</p>
```

Record Type

Gemeinsames Beispiel

$(4 + 6i) + (8 + -3i) = 12 + 3i$

$(4 + 6i) - (8 + -3i) = -4 + 9i$

$(4 + 6i) * (8 + -3i) = 50 + 36i$

$(4 + 6i) / (8 + -3i) = 0.1917808219178082 + 0.821917808219178i$

Ausgewählte Techniken 7/26

- Deklaration von wiederverwendbaren eigenen Typen – `xsl:item-type` (1)
 - ✓ Im Beispiel Komplexe Zahlen mit `record` 1x anlegen und mit `type(...)` mehrfach anwenden

```
<xsl:item-type name="complex" as="record(r as xs:double, i as xs:double)"/>
```

```
<xsl:function name="cx:complex" as="type(complex)">...</xsl:function>
```

```
<xsl:function name="cx:add" as="type(complex)">...</xsl:function>
```

```
<xsl:function name="cx:sub" as="type(complex)">...</xsl:function>
```

```
<xsl:function name="cx:mul" as="type(complex)">...</xsl:function>
```

```
<xsl:function name="cx:div" as="type(complex)">...</xsl:function>
```


Ausgewählte Techniken 8/26

- Deklaration von wiederverwendbaren eigenen Typen – `xsl:item-type` (2)
 - ✓ Idee für einen geometrischen Point Type als record

```
<xsl:item-type name="point_record" as="record(x as xs:double, y as xs:double)"/>
```

```
<xsl:function name="tm:point" as="type(point_record)">  
  <xsl:param name="x" as="xs:double"/>  
  <xsl:param name="y" as="xs:double"/>  
  <xsl:sequence select="map{'x':$x, 'y':$y}"/>  
</xsl:function>
```

<!-- Nutzung in einer Funktion für den Abstand zwischen zwei Punkten -->

```
<xsl:function name="tm:distance" as="xs:double">  
  <xsl:param name="point1" as="type(point_record)"/>  
  <xsl:param name="point2" as="type(point_record)"/>  
  <xsl:sequence select="math:sqrt(math:pow($point1?x - $point2?x, 2) + math:pow($point1?y - $point2?y, 2))"/>  
</xsl:function>
```

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Ausgewählte Techniken ^{9/26}

- Deklaration von wiederverwendbaren eigenen Typen – `xsl:item-type` (3)

✓ Idee für einen geometrischen Point Type als record

```
<!-- Anwendung mit SVG - aus zwei Punkten wird Gerade erzeugt und mit Abstand ausgegeben -->
<xsl:variable name="p1" select="tm:point(75, 50)" as="type(point_record)"/>
<xsl:variable name="p2" select="tm:point(300, 150)" as="type(point_record)"/>
<xsl:variable name="abstand" select="tm:distance($p1, $p2)" as="xs:double"/>

<svg xmlns="http://www.w3.org/2000/svg" width="500" height="250">
  <text x="0" y="30" font-size="20">Abstand zwischen zwei Punkten: {fn:round($abstand, 2)}</text>
  <line x1="{ $p1?x }" y1="{ $p1?y }" x2="{ $p2?x }" y2="{ $p2?y }" stroke="#F00" stroke-width="2"/>
  <circle cx="{ $p1?x }" cy="{ $p1?y }" r="2" fill="#00F"/>
  <circle cx="{ $p2?x }" cy="{ $p2?y }" r="2" fill="#00F"/>
</svg>
```

xsl:item-type mit record()

Abstand zwischen zwei Punkten: 246.22



Ausgewählte Techniken 10/26

- Neue Attribute then / else für xsl:if
 - ✓ Ermöglicht inzeilige Abfragen im Element-Kontext (analog zu if / then / else für Attribute)
 - ✓ Alternative Nutzung als Rückgabewert in xsl:function

```
<xsl:variable name="tag" select="fn:format-date(fn:current-date(), '[F]')"/>
<p>
  {$tag} | <xsl:if test="$tag = 'Mittwoch'" then="23" else="42"/>
</p>

<figure>
  
  <figcaption>
    Let's drink SAXON-<xsl:if
      test="fn:current-time() ge xs:time('18:00:00')"
      then="'Gin'" else="'Water'"/>!
  </figcaption>
</figure>
```

xsl:if

Mittwoch | 23



Let's drink SAXON-Gin!



Let's drink SAXON-Water!

Ausgewählte Techniken 11/26

- Neues select-Attribut für xsl:when und xsl:otherwise (unter xsl:choose)

✓ Direkte Wertzuweisung bei der Abfrage statt im Element-Kontext

```
<!-- Einsatz innerhalb einer eigenen Funktion -->
<xsl:variable name="tag" select="fn:format-date($datum, '[F]')"/>
<xsl:variable name="mi1" as="xs:integer">
  <xsl:choose>
    <xsl:when test="$tag = 'Mittwoch'" select="1"/>
    <xsl:when test="$tag = 'Dienstag'" select="2"/>
    <xsl:when test="$tag = 'Montag'" select="3"/>
    <xsl:when test="$tag = 'Sonntag'" select="4"/>
    <xsl:when test="$tag = 'Samstag'" select="5"/>
    <xsl:when test="$tag = 'Freitag'" select="6"/>
    <xsl:when test="$tag = 'Donnerstag'" select="7"/>
    <!-- ggf. <xsl:otherwise select="..." /> -->
  </xsl:choose>
</xsl:variable>

<xsl:variable name="mi3" select="$mi1 + 14"/>
<xsl:value-of select="$mi3 || '.' || $monate[$i] || '.' || $jahr"/>
```

Termine 2022

3. Mittwoch im Monat:

1. 19.01.2022
2. 16.02.2022
3. 16.03.2022
4. 20.04.2022
5. 18.05.2022
6. 15.06.2022
7. 20.07.2022
8. 17.08.2022
9. 21.09.2022
10. 19.10.2022
11. 16.11.2022
12. 21.12.2022

Ausgewählte Techniken ^{12/26}

- Neues Element für konditionale Abfragen – xsl:switch

✓ Erweitert die Möglichkeiten analog zu anderen Sprachen

```
<xsl:variable name="tekomp" select="xs:date('2022-11-08'), xs:date('2022-11-09'), xs:date('2022-11-10')" as="xs:date*" />
<xsl:variable name="datum" select="xs:date('2022-11-09')" as="xs:date" /> <!-- hier fn:current-date() abfragen -->
<xsl:variable name="check" select="fn:index-of($tekomp, $datum)" as="xs:integer*" />
```

```
<p>
  <xsl:switch select="if(fn:exists($check)) then $check else 0">
    <xsl:when test="1">1. Tag der tekomp-Jahrestagung.</xsl:when>
    <xsl:when test="2">2. Tag der tekomp-Jahrestagung.</xsl:when>
    <xsl:when test="3">3. Tag der tekomp-Jahrestagung.</xsl:when>
    <xsl:when test="0">Ein anderer Tag.</xsl:when>
    <!-- oder statt 0-Prüfung: -->
    <!-- <xsl:otherwise>Ein anderer Tag.</xsl:otherwise> -->
  </xsl:switch>
</p>
```

xsl:switch

2. Tag der tekomp-Jahrestagung.

Ausgewählte Techniken 13/26

- Variable Argumentanzahl (Arity) für xsl:function

✓ Bisher nur durch mehrfache Deklaration einer Funktion möglich

```
<xsl:function name="tm:addition" as="xs:decimal">
  <xsl:param name="a" as="xs:decimal"/>
  <xsl:param name="b" as="xs:decimal"/>
  <xsl:param name="c" as="xs:decimal"/>
  <xsl:param name="d" as="xs:decimal"/>
  <xsl:sequence select="$a + $b + $c + $d"/>
</xsl:function>
```

```
<xsl:function name="tm:addition" as="xs:decimal">
  <xsl:param name="a" as="xs:decimal"/>
  <xsl:param name="b" as="xs:decimal"/>
  <xsl:sequence select="$a + $b"/>
</xsl:function>
```

✓ Neue Attribute `required` und `select` (aktuell noch nicht unterstützt)

```
<xsl:function name="tm:addition" as="xs:decimal">
  <xsl:param name="a" as="xs:decimal"/>
  <xsl:param name="b" as="xs:decimal"/>
  <xsl:param name="c" as="xs:decimal" required="no" select="0"/>
  <xsl:param name="d" as="xs:decimal" required="no" select="0"/>
  <xsl:sequence select="$a + $b + $c + $d"/>
</xsl:function>
```

```
<p>{tm:addition(23,42)}</p>
<p>{tm:addition(1,2,3,4)}</p>
```

xsl:function

65

10

- Neues Element `xsl:function-library`
 - ✓ Lässt eine Bibliothek vermuten, kapselt jedoch nur Namespaces in externen Modulen und ermöglicht bei Bedarf den Ausschluss bestimmter Funktionen (noch nicht testbar)

```
<xsl:function-library>
  <xsl:function-namespace prefix="fn"      uri="http://www.w3.org/2005/xpath-functions"/>
  <!-- exclude="count" -->
  <xsl:function-namespace prefix="math"    uri="http://www.w3.org/2005/xpath-functions/math"/>
  <!-- exclude="sin cos" -->
  <xsl:function-namespace prefix="map"     uri="http://www.w3.org/2005/xpath-functions/map"/>
  <xsl:function-namespace prefix="array"   uri="http://www.w3.org/2005/xpath-functions/array"/>
  <xsl:function-namespace prefix="xs"      uri="http://www.w3.org/2001/XMLSchema"/>
  <xsl:function-namespace prefix="tm"      uri="http://www.datenverdrahten.de/tm"/>
</xsl:function-library>
```

Ausgewählte Techniken 15/26

- Neues Element `xsl:match`

✓ Prüfung einer Bedingung, etwa in der Dokumentstruktur → true | false

```
<xsl:variable name="doc" select="fn:doc('test.xml')" as="document-node()"/>

<xsl:variable name="check" as="xs:boolean">
  <xsl:match select="$doc/root/*" pattern="element1[@attr='abc'] | element2"/>
</xsl:variable>

<p>Prüfung <xsl:if test="$check" then="'ok'" else="'nicht ok'"/>.</p>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <element1 attr="abc">Text 1</element1>
  <element2>Text 2</element2>
</root>
```

xsl:match

Prüfung ok.

- Attribut separator

✓ Für xsl:apply-templates und xsl:for-each ergänzt, existiert bereits für xsl:value-of

```
<xsl:template match="test">
  <xsl:value-of select="."/>
</xsl:template>
```

```
<p>
  <xsl:apply-templates select="fn:doc('separator_attribute.xml')/root/test" separator=" | ">
</p>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <test>1</test>
  <test>2</test>
  <test>3</test>
  <test>4</test>
  <test>5</test>
</root>
```

Separator

1|2|3|4|5

Ausgewählte Techniken 17/26

- Erweiterung für benannte Templates
 - ✓ Beispiel zur mehrfachen inzeiligen Ausgabe von Zeichen

```
<xsl:template name="ex:plus">
  <xsl:param name="p" as="xs:integer"/>
  <xsl:value-of select="fn:string-join((1 to $p) ! '+')"/>
</xsl:template>
```

```
<p>Text <ex:plus p="3"/> Text <ex:plus p="4"/> Text <ex:plus p="5"/> Text</p>
```

```
<h2><ex:plus p="3"/> Eilmeldung <ex:plus p="3"/> XSLT 4.0 in Sicht! <ex:plus p="3"/></h2>
```

Erweiterung benannter Templates

Text +++ Text ++++ Text +++++ Text

+++ Eilmeldung +++ XSLT 4.0 in Sicht! +++

- Techniken für Arrays und Maps (1)

✓ xsl:array / xsl:array-member (analog zu xsl:map / xsl:map-entry)

```
<xsl:variable name="q_zahlen" as="array(*)">
  <xsl:array composite="yes">
    <xsl:for-each select="1 to 10">
      <xsl:array-member>{. * .}</xsl:array-member>
      <!-- oder: <xsl:array-member select="." * ."/> -->
    </xsl:for-each>
  </xsl:array>
</xsl:variable>
```

```
<p>{$q_zahlen}</p>
<p>{$q_zahlen(5)}</p>
```

xsl:array / xsl:array-member

1 4 9 16 25 36 49 64 81 100

25

- Techniken für Arrays und Maps (2)

- ✓ Array-Iteration mit for member

for member

abcde | xyz

```
<!-- Beispiel mit Zeichen(ketten) -->
<xsl:variable name="b_arr" select="['e','b','c','a','d'], ('z','x','y')" as="array(*)"/>
<xsl:variable name="b_seq" select="for member $m in $b_arr return fn:string-join(fn:sort($m), ' ')"
  as="xs:string*"/>
<p><xsl:value-of select="$b_seq" separator=" | "/></p>
```

```
<!-- Beispiel mit Zahlenwerten -->
<xsl:variable name="d_arr" select="[(1,2,3,4,5), (10,20,30), (23,42)]" as="array(*)"/>
<xsl:variable name="s_seq" select="for member $m in $d_arr return fn:sum($m)" as="xs:decimal*"/>
<xsl:variable name="a_seq" select="for member $m in $d_arr return fn:avg($m)" as="xs:decimal*"/>
```

```
<table>[...Kopf...]<xsl:for-each select="1 to array:size($d_arr)">
  <xsl:variable name="i" select="fn:position()" as="xs:integer"/>
  <tr><td><xsl:value-of select="($d_arr($i))" separator=","/></td>
    <td>{$s_seq[$i]}</td><td>{$a_seq[$i]}</td></tr>
</xsl:for-each>
```

```
</table>
```

Daten	Summe	Mittelwert
1,2,3,4,5	15	3
10,20,30	60	20
23,42	65	32.5

- Techniken für Arrays und Maps (3)

✓ Iteration über Arrays und Maps (?value / ?key = lookup expressions)

```
<p>
  <xsl:for-each array="[3, 9, 1, 7, 5]" separator=" | ">
    <xsl:sort select="?value" order="ascending"/> <!-- value-Sortierung bei Arrays -->
    {?value * 2}
  </xsl:for-each>
</p>
```

Iteration über Arrays und Maps

```
<p>
  <xsl:for-each array="[1 to 5, 9 to 17, 1 to 100]" separator=" | ">
    {fn:sum(?value)}
  </xsl:for-each>
</p>
```

2 | 6 | 10 | 14 | 18

15 | 117 | 5050

123 | 456 | 789

```
<p>
  <xsl:for-each map="map{'a': 789, 'b': 456, 'c': 123}" separator=" | ">
    <xsl:sort select="?key" order="descending"/> <!-- key/value-Sortierung bei Maps -->
    {?value}
  </xsl:for-each>
</p>
```


Ausgewählte Techniken 21/26

- Techniken für Arrays und Maps (4)

- ✓ Attribut on-duplicates für xsl:map

```
<xsl:template match="data">
  <xsl:map on-duplicates="function($a, $b){array:join(($a, $b))}">
    <xsl:for-each select="event">
      <xsl:map-entry key="@id" select="[fn:number(@value)]"/>
    </xsl:for-each>
  </xsl:map>
</xsl:template>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<data>
  <event id="A23" value="12"/>
  <event id="A24" value="5"/>
  <event id="A23" value="7"/>
  <event id="A25" value="9"/>
  <event id="A23" value="2"/>
</data>
```

JSON-Ausgabe

```
{
  "A23": [ 12, 7, 2 ],
  "A24": [ 5 ],
  "A25": [ 9 ]
}
```

- Weitere Funktionen und Techniken von XPath 4.0 (1)

- ✓ **fn:isNaN (is Not a Number)**

`fn:isNaN(123) → false`

`fn:isNaN(math:sqrt(-1)) → true`

- ✓ **fn:characters (splittet String in Sequenz aus Zeichen)**

`fn:characters("Hallo Welt!") → ("H", "a", "l", "l", "o", " ", "W", "e", "l", "t", "!")`

- ✓ **fn:slice (Zeichenauswahl anhand ihrer Position)**

`fn:slice(("a", "b", "c", "d", "e"), start:2, end:4) → ("b", "c", "d")`

- ✓ **fn:some (mindestens ein Item der Sequenz erfüllt Prädikat)**

`fn:some((1, 2, 3), ->{. mod 2 = 0}) → true`

`fn:some(("Januar", "Februar", "März", "April", "Mai", "Juni", "Juli", "August", "September", "Oktober", "November", "Dezember"), contains(?, "n")) → true`

- ✓ **fn:all (alle Items der Sequenz erfüllen Prädikat)**

`fn:all((1, 2, 3), ->{. mod 2 = 0}) → false | fn:all(...) für Monate mit n → false`

- Weitere Funktionen und Techniken von XPath 4.0 (2)

- ✓ **fn:range** (Bereich abfragen – in der Spec *fn:range-from* / *fn:range-to*)

`fn:range(("a", "b", "c", "d", "e"), matches(?, "c"))` → ("c", "d", "e")

- ✓ **fn:highest** (Items einer Sequenz mit höchsten Werten zur Bedingung)

`fn:highest(1 to 25, (), ->{. idiv 10})` → 20 bis 25 bezogen auf Ergebnis 2

- ✓ **fn:lowest** (Items einer Sequenz mit niedrigsten Werten zur Bedingung)

`fn:lowest(1 to 25, (), ->{. idiv 10})` → 1 bis 9 bezogen auf Ergebnis 0

- ✓ **fn:index-where** (Position der Items in der Sequenz, welche Bedingung erfüllen)

`fn:index-where(1 to 10, ->{. mod 2 = 0})` → 2 4 6 8 10

`fn:index-where(("blau", "gelb", "rot", "schwarz"), ->{fn:contains(., "a")})` → 1 4

Ausgewählte Techniken 24/26

- Weitere Funktionen und Techniken von XPath 4.0 (3)
 - ✓ `fn:items-after` / `fn:items-before` / `fn:items-from` / `fn:items-until`
→ Teilsequenzen anhand des Funktionsergebnisses `true` | `false`

```
<xsl:variable name="s" select="('a', 'ab', 'abc', 'xyz')" as="item()*"/>
```

```
fn:items-after($s, function($x){fn:string-length($x) = 1}) → ("ab", "abc", "xyz")
```

```
fn:items-before($s, function($x){fn:string-length($x) = 2}) → ("a")
```

```
fn:items-from($s, function($x){fn:string-length($x) = 3}) → ("abc", "xyz")
```

```
fn:items-until($s, function($x){fn:string-length($x) = 3}) → ("a", "ab", "abc")
```

Ausgewählte Techniken ^{25/26}

- Weitere Funktionen und Techniken von XPath 4.0 (4)

- ✓ Ternary Conditions

```
<xsl:variable name="w" as="xs:string" select="23 lt 42 ?? 'ok' !! 'nicht ok'"/>
<p>{$w}</p> <!-- ok -->
```

- ✓ Otherwise operator

```
<xsl:variable name="a" as="xs:positiveInteger*" select="10"/>
<!-- <xsl:variable name="b" as="xs:positiveInteger*" select="5"/> -->
<xsl:variable name="b" as="xs:positiveInteger*" select="()" />
<p>{$a div $b otherwise 2}</p> <!-- bedeutet: $a div ($b otherwise 2) | Ergebnis: 2 bzw. 5 -->
```

- ✓ Simple inline functions

```
<xsl:variable name="f" select="function($x, $y){$x + $y}"/>
<p>{$f(23, 42)}</p> <!-- 65 -->
<xsl:variable name="g" select="->($x, $y){$x + $y}"/> <!-- Thin arrow notation -->
<p>{$g(23, 42)}</p> <!-- 65 -->
```


Ausgewählte Techniken 26/26

- Weitere Funktionen und Techniken von XPath 4.0 (5)

- ✓ Mathematical Operator Symbols

→ hier für kleiner als \leq statt lt (noch nicht unterstützt)

```
<xsl:variable name="w" as="xs:string">
  <!-- <xsl:if test="23 < 42" then="'ok'" else="'nicht ok'"/> -->
  <xsl:if test="23 lt 42" then="'ok'" else="'nicht ok'"/>
</xsl:variable>

<p>{$w}</p> <!-- ok -->
```

Operator	Symbol	Codepoint
and	\wedge	x2227
or	\vee	x2228
eq	\equiv	x2250
ne	\neq	x2260
lt	\leq	x22D6
gt	\geq	x22D7
le	\leq	x2264
ge	\geq	x2265
div	\div	xF7
mod		
idiv	\oplus	x2A38
union ()	\cup	x222A
intersect	\cap	x2229
except	\setminus	x2216
is	\equiv	x2261
<< (precedes)	\ll	x226A
>> (follows)	\gg	x226B
otherwise	\Vdash	x22A9
some	\exists	x2203
every	\forall	x2200
satisfies	\Rightarrow	x29F4

Fazit und Ausblick

- Neue Möglichkeiten bereichern die Programmierpraxis je nach konkreten Anforderungen.
- Erweiterungen für Arrays und Maps zielen deutlich in Richtung alternativer Verarbeitung, speziell von JSON-Daten.
- Als praktisch erweisen sich auch die kleineren Ergänzungen, etwa die Attribute `then / else` für `xsl:if`, das Attribut `select` bei `xsl:when / xsl:otherwise` sowie das Element `xsl:switch`.
- Der neue `record`-Typ bietet interessante Ansätze für eigene Datenstrukturen und neue Ideen.
- Bleibt zu hoffen, dass die W3C-Arbeitsgruppe vielleicht 2023 – also drei Jahre nach dem Proposal von Michael Kay – die 4.0-Spezifikationen finalisiert und diese durch Software optimal unterstützt werden.

Danke für die Aufmerksamkeit!



seasonally affected server

@sadsrver

Failure is an important part of the learning process.

I'm sure you'll be learning a lot this week.

[Tweet übersetzen](#)

6:39 nachm. · 29. Aug. 2022 · TweetDeck

Bewertung:

