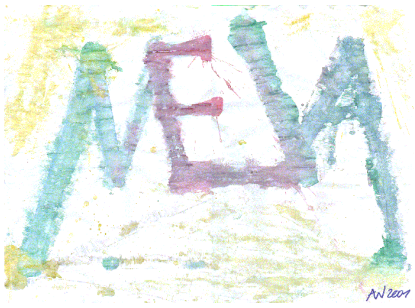
Albrecht Weinert

# XSLT 1.0 Tip

## Implementing a first character range filter



Last mod.: 19.02.2013

Prof. Dr.-Ing. Albrecht Weinert a-weinert.de
weinert – automation weinert-automation.de

Labor für Medien und verteilte Anwendungen (MEVA-Lab) meva-lab.de
Fachbereich Informatik der Hochschule Bochum

# Implementing a first character range filter with pure XSLT 1.0

V01.00, 16.12.2012: new
V01.01, 18.02.2013: minor spelling corrections
V01.02, 16.12.2013: -

Version: V1.01
Last modified by A. Weinert at 19.02.2013



Note on the template: There is one common numbering for figures, lists, tables etc. (if present)

Note on version control: SVN URL is https://ai2t.de/svn/albrecht/pub/XSLT1.0RangeFilter.odt

Note on publications: See also http://a-weinert.de/publication_en.html ,
http://blog.a-weinert.de/ and
http://blog.a-weinert.de/allgemein/character-range-filter-with-xslt-1-0/ .

This document's URL: http://a-weinert.de/pub/XSLT1.0RangeFilter.pdf.
The version there might be newer if this is from elsewhere or on paper.

## Table of content

1. Scope ....2
2. Problem ....2
2.1 Still using XSLT 1.0 for good reasons ....2
2.2 XSLT 1.0's abashing facilities for string / character operations ....2
3. Solution – the 1.0 character range filter ....3
4. Resume ....4
A Abbreviations ....4

# 1. Scope

This is a tip about implementing a first character range filter with pure XSLT 1.0.

Though being a bit out of the main hype, XML data files and transforming style sheets are still attractive to separate data and rendering.
So this might be helpful for regular xml / xls users / writers.

# 2. Problem

The transformers and and their description, existing in version XSLT 2.0, are quite powerful, as soon as being accustomed to xls scatterbrained syntax. XSLT 1.0 is, of course, in no way better and much less expressive and powerful.

## 2.1 Still using XSLT 1.0 for good reasons

But there are cases, where one has to stick to XSLT 1.0 without any extensions. This would be mainly due to some server's / framework's / PHP installation's and client's limitations. On those one often has little to no influence. Using XSLT 2.0 or any extensions not covered by the standard and then hitting one of the above said limited environments in deployment / at customer sites is a way to to get into deep doo-doo.

In other words it may sometimes still be clever to stick to pure XSLT 1.0 against all advices and (what holds you off 2.0?) mockery.

## 2.2 XSLT 1.0's abashing facilities for string / character operations

One worst limitations of XSLT 1.0 (and hence one of the highest prices to pay) is the almost total absence of decent string and character functions respectively operators. At one hand it is possible to sort using a string attribute (or value) as key.

```
<xsl:for-each
   select="document('members_lists.xml')//member-lists//personen//person">
     <!-- sn is of type xsd:string         -->
    <xsl:sort select="@sn" order="ascending" />
```

Listing 1: Sorting by a key of type string.

Obviously it's often required to limit respectively filter the iteration to a range of the same (sorting) key, say “Meyer .. Möller” for example. Having no big problem with that in XSLT 2.0, all attempts usually fail in 1.0. Reducing the requirement to the more popular first letter range, say 'J..Ö' for example doesn't make it really better. And by the way, all tips found either fail to implement the range filter or leave pure XSLT 1.

Base of problem is XSLT 1.0 being able to compare strings and characters by operators (= and !=) for equality but unable to (lexically) compare them. Background and starting point of all troubles is that the operators >=, <= and all comparison operators, except != and = only treat numbers [sic!]. Hence, as most strings (“Möller”, “Meyer”, “J”, “Ö”) automatically convert to NaN (not a number) all comparisons will return false – conforming to IEEE 754.

So use string comparison functions instead of operators? Nil report! There are none in XSLT 1.0. Believe it! Well, it is beyond belief, as the sorting of Listing 1 is perfectly done in XSLT 1.0 even, as in above examples, with Umlauts and else.
So 1.0 has perfect character and string comparison implementations. It, inconceivably, just hides them from us.

## 3. Solution – the 1.0 character range filter

At a point in the .xsl-file global enough for all usages (mostly for-each blocks) we define the alphabet of all characters allowed in our intended (filter) order, as in Listing 2

```
<xsl:variable name="sortChar">
    "0123456789AÄaäBbCcDdEeFfGgHhIiJjKkLlMmNnOÖoöPpQqRrSsßTtUÜuüVvWwXxYyZz"
 </xsl:variable>
```

Listing 2: Defining the filtering alphabet and order.

At the point we have the actual range setting (as two string attributes `from` and `toBefore` in the example) we insert Listing 3.

```
<xsl:variable name="frmFrst" >
    <xsl:value-of select="substring(@from, 1,1)"/></xsl:variable>
<xsl:variable name="befFrst">
    <xsl:value-of select="substring(@toBefore, 1,1)"/></xsl:variable>

<xsl:variable name="frmCmp">
    <xsl:value-of select="string-length(
        substring-before($sortChar, $frmFrst))" /></xsl:variable>
<xsl:variable name="befTmp"><xsl:value-of select="string-length(
        substring-before($sortChar, $befFrst))" /></xsl:variable>
<xsl:variable name="befCmp"><xsl:choose>
    <xsl:when test="$befTmp = 0">99</xsl:when>
    <xsl:otherwise><xsl:value-of select="$befTmp"/></xsl:otherwise>
</xsl:choose></xsl:variable>
```

Listing 3: Getting a number range from two strings (first character of each).

For the range start character `frmFrst` got as first character of attribute `@from` in the example we get its position number within our filter alphabet defined in Listing 2. A character not defined there or no character at all gets 0. The meaning will be “from any” character respectively “to any” for a filter according to Listing 4. For the (exclusive) upper bound we get the number befCmp by the same algorithm, additionally setting a high value for undefined or no character to get said “to any” behaviour.

```
<xsl:variable name="idFrst">
   <xsl:value-of select="substring(@sn, 1,1)"/></xsl:variable>
<xsl:variable name="idCmp"><xsl:value-of
   select="string-length(substring-before($sortChar, $idFrst))" />
</xsl:variable>
<xsl:if test="$idCmp >= $frmCmp and $befCmp > $idCmp">
```

Listing 4: The range filter.

Finally for the for the filter we add something like Listing 4 immediately after the snippet of listing 1, i.e. the respective iteration. Here we apply the same processing to the string value in question, that is the attribute `@sn` in the example. Having now three integer numbers (`$frmCmp`, `$idCmp` and `$befCmp` in the example) as lower bound, value and (exclusive) upper bound the filter condition is trivially expressed with XSLT 1.0 operators.

## 4. Resume

Well, the algorithm is not beautiful, to put it mildly. One can even say it is outright ugly and not quite resource saving, but it works perfectly at different places. The string functions used in above exemplary listing are provided by XSLT 1.0.

Especially we have no indexOf, toUpperCase or toLowerCase – the latter two habitually being done by translate and two strings à la “ABCD...”, “abcd...”. So there seams little way to express above range filtering with borders given in a .xml more compact.

## A Abbreviations

| | |
|---|---|
| HTML | Hypertext Markup Language [RFC 1866] |
| HTTP | Hypertext Transfer Protokoll |
| HTTPS | HTTP via SSL |
| JAXP | Java API for XML Parsing |
| SSL | Secure Socket Layer |
| XML | eXtensible Markup Language |
| XSD | XML schema definition |
| XSL | eXtensible Stylesheet Language |
| XSLT | XSL Transformation |