Using XSLT for getting back-of-the-book indexes

Jirka **Kosek** <jirka@kosek.cz>

Abstract

Many electronic publishing systems built on the top of XML (e.g. DocBook) use XSLT to convert source XML document into target formats like HTML or XSL-FO (for print output). During the transformation back-of-the-book index can be generated and populated by index entries spread over the document. Creating index basically means to sort and group index entries by their first letter. However this solutions is appropriate only for some languages, English included. For other Latin based languages like Czech, Hungarian or Spanish grouping method is more sophisticated and can't be expressed in the standard XSLT 1.0. The task is even more challenging if we want to get internationalized indexes in some general stylesheet package like DocBook XSL stylesheets. These stylesheets should support as many XSLT implementations as possible what disqualifies usage of vendor extensions.

This paper will show you how support for non-English index generation was implemented in the DocBook XSL stylesheets, what problems were overcame and what functionality is missing in XSLT 1.0, but can be added using EXSLT extensions. To deal with grouping problems like different accented letters belonging to the same group, multi-letter sequences denoting one group etc. solution based on XSLT keys over user defined function is provided. This function uses external localization files to lookup values which drive index generation and grouping.

Method presented up to this point is sufficient for indexes in HTML output. Print output brings new problems. As the transformation and formatting phases in the XSL are separated there is no direct support for merging duplicate page numbers in XSL-FO. Fortunately many FO engine vendors provide custom extensions to deal with this issue. Integration of these extensions into the DocBook XSL stylesheet will be presented.

Article also includes evaluation of XSLT 2.0 features available for index generation and proposals for further improvement of indexing method that will be able to handle CJKV languages.

Table of Contents

XSL•FO
RenderX

1. Introduction

Usability of a document, especially a printed document, depends on a good back-of-the-book index. Creating an index is a very laborious and responsible work often performed by specialists. At these days indexes are not built manually after the final layout of book is known, instead index terms are marked directly in a manuscript and index is then built automatically during the document formatting.

Nowadays in an era of XML publishing several document types suitable for large documents emerged. Probably the best known and used is DocBook [http://docbook.org]. DocBook DTD defines several elements for marking up index terms. These terms are properly formed into an index when the document is processed by the DocBook XSL stylesheets. However getting proper index by means of XSLT 1.0 and XSL-FO 1.0 is almost impossible. In the following text I'm going to show you where are the limits of current XSL regarding the index generation, how they were overcome in the DocBook stylesheets and how new versions of XSLT and XSL-FO will make this task easier.

2. Marking up index entries in the DocBook

The most difficult part of creating an index must be done manually and consists of marking up index entries in a document. In DocBook, this is done by placing the `indexterm` elements everywhere where you write about the given topic. The content of the `indexterm` is not displayed as a part of a document flow; it is used later when building the index.

```
<para>Wealth of a modern societies is built upon
information<indexterm><primary>information</primary></indexterm>.</para>
```

The `indexterm` element can also hold multilevel entries:

```
<indexterm>
<primary>information</primary>
</indexterm>
...
<indexterm>
<primary>information</primary>
<secondary>retrieval</secondary>
</indexterm>
...
<indexterm>
<primary>information</primary>
<secondary>dissemination</secondary>
</indexterm>
...
<indexterm>
<primary>information</primary>
<secondary>dissemination</secondary>
<tertiary>oral</tertiary>
</indexterm>
```

Such index terms will result into the following index output (the page numbers are of course for illustration only):

```
information, 13
  dissemination, 17
    oral, 25
  retrieval, 15
```

DocBook markup offers several other more advanced methods for marking up index entries. You can read about them in [TDG] or [DBIDX]. In the following text I will assume only single level entries because multilevel entries do not add any significant processing complexity from the internationalization point of view.

3. Using XSLT to generate index

Generating the index consists of grouping the index terms with the same initial letters and then alphabetical sorting of the entries within each letter group. The stylesheets exactly implement this algorithm.

The most common design pattern for grouping in XSLT is so called Muenchian method. In order to use this method we must define a key that indexes all elements to be grouped based on their group key. This means that we must create key which will cover all `indexterm` elements and key will be based on the first letter of an index term.

```
<xsl:key name="letter" match="indexterm" use="substring(primary,1,1)"/>
```

Moreover we want to group index terms regardless of their case. Thus key should be created over lower-cased or upper-cased letters. DocBook stylesheets use later approach. The final definition of key is

```
<xsl:key name="letter" match="indexterm"
         use="translate(substring(primary,1,1),
                        'abcdefghijklmnopqrstuvwxyz',
                        'ABCDEFGHIJKLMNOPQRSTUVWXYZ')"/>
```

We need to select each letter used as the first letter of an index term just once. We do it by selecting first index term starting with such letter and processing it in a special mode that is responsible for producing one index group.

```
<xsl:apply-templates select="//indexterm[count(.|
                                key('letter',
                                  translate(substring(primary,1,1),
                                    'abcdefghijklmnopqrstuvwxyz',
                                    'ABCDEFGHIJKLMNOPQRSTUVWXYZ'))[1])
                                = 1]"
                     mode="index-div">
  <xsl:sort select="translate(primary, 'abcdefghijklmnopqrstuvwxyz',
                                       'ABCDEFGHIJKLMNOPQRSTUVWXYZ')"/>
</xsl:apply-templates>
```

When processing an index group we must emit the group label and process all entries belonging to this particular group. Entries in the group must be also sorted and grouped together. For this purpose second `xsl:key` is defined and Muenchian method is used once more.

```
<xsl:key name="primary" match="indexterm" use="primary"/>

<xsl:template match="indexterm" mode="index-div">
  <!-- Get the group key (ie. first letter of index terms in this group -->
  <xsl:variable name="key"
```

```
                select="translate(substring(primary,1,1),
                                  'abcdefghijklmnopqrstuvwxyz',
                                  'ABCDEFGHIJKLMNOPQRSTUVWXYZ')"/>

  <!-- Output label for current index group -->
  <xsl:value-of select="$key"/>

  <xsl:apply-templates select="key('letter', $key)
                             [count(.|key('primary',primary)[1])=1]"
                       mode="index-primary">
    <xsl:sort select="translate(primary,
                                'abcdefghijklmnopqrstuvwxyz',
                                'ABCDEFGHIJKLMNOPQRSTUVWXYZ')"/>
  </xsl:apply-templates>
</xsl:template>
```

For each entry in the group the following template is called. The template outputs index term and adds all page references after the term. If there are any secondary terms defined they are processed then.

```
<xsl:template match="indexterm" mode="index-primary">

  <!-- Find all occurences of index term -->
  <xsl:variable name="refs" select="key('primary', primary)"/>

  <!-- Output text of index term -->
  <xsl:value-of select="primary"/>

  <xsl:for-each select="$refs[not(secondary)]">
    <!-- Create page number reference (print) or link with back
         reference (HTML) to each occurrence of the index term -->
  </xsl:for-each>

  <xsl:if test="$refs/secondary">
     <!-- Process secondary level entries -->
  </xsl:if>
</xsl:template>
```

As we can see indexing code is not very complex if one knows tricks like Muenchian grouping or method to implement upper case function. The actual code in the DocBook XSL stylesheets is more complex as it deals with all ways in which index terms can be expressed. But principal method is the same.

The described method is able to generate satisfactory back-of-the-book index for English documents. But it is clear that for other languages is inappropriate. Accented letters are not grouped together with unaccented ones.[1] Other Latin based languages have separate groups for letters composed from two characters.[2] The biggest problem is that these rules are unique for each language and as such should be localized.

[1]For example in German letter “ö” should be in the same group as “o” and within this group it should be sorted as “oe”.
[2]For example in Czech “ch” is treated as one letter, it should have separate group in index which is placed between “h” and “i”.

4. DocBook stylesheets and localization

The DocBook XSL stylesheets adapt its output to a document language. This means that automatically generated texts like “Table of Contents”, “Figure” or shape of quotes marks are different for each language. The document language can be specified by using the `lang` attribute.

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE book PUBLIC '-//OASIS//DTD DocBook XML V4.3//EN'
          'http://www.oasis-open.org/docbook/xml/4.3/docbookx.dtd'>
<book lang="de">
  ... German book ...
</book>
```

How does the DocBook localization work? In the stylesheets distribution you can find many files named like `en.xml`, `cs.xml` or `pt_br.xml` in the `common` directory. Files are named by coressponding ISO 639 language code. Each of these files contain translations of text phrases into the target language. At the time of this writing stylesheets supported 44 different locales. The following fragment shows how the translation of phrase “Table of Contents” into Russian is recorded in the localization file.

```
<l:gentext key="TableofContents" text="          " />
```

All locale files are merged into one XML document `l10n.xml` using entities.[3] This document is then used to lookup translations or if there is no corresponding translation then to get default English value. Code that deals with finding correct translation is placed in `l10n.xsl` stylesheet. The two most important templates here are named templates `gentext` and `gentext.template`. Their usage is very simple. If you want to get translation of “Table of Contents” in a current document language you can simply call the `gentext` template with appropriate parameter.

```
<xsl:call-template name="gentext">
  <xsl:with-param name="key">TableofContents</xsl:with-param>
</xsl:call-template>
```

It seems natural to use the same localization mechanism to get localized index. For many Latin based languages it should be sufficient to treat some accented characters like unaccented in order to place them into the same index group. This can be accomplished by changing parameters of `translate()` function which is used to convert index terms to uppercase during grouping. For example in German umlauted characters should be grouped as if they are without umlaut. This can be done by using the following parameters for `translate()` in XSLT indexing code.

```
translate(substring(primary,1,1),
          'abcdefghijklmnopqrstuvwxyzäÄöÖüÜ',
          'ABCDEFGHIJKLMNOPQRSTUVWXYZAAOOUU')
```

Instead of original

```
translate(substring(primary,1,1),
          'abcdefghijklmnopqrstuvwxyz',
          'ABCDEFGHIJKLMNOPQRSTUVWXYZ')
```

[3]Merged document is quite large and has about two megabytes. If you are really concerned about performance you can comment out entity references for unused languages in `l10n.xml`. Such information arrangement may be seen as a very ineffective but please have in mind that it was designed several years ago when there were just few localizations and there were also much less items to translate.

XSL•FO
RenderX

The problem is that we can not call `gentext` template directly from XPath expression because XSLT named templates are not treated as XPath extension functions. Instead we must grab result of calling template into a variable and then use this variable in an expression.

```
<xsl:variable name="index.lowercase">
  <xsl:call-template name="gentext">
    <xsl:with-param name="key">index.lowercase</xsl:with-param>
  </xsl:call-template>
</xsl:variable>

<xsl:variable name="index.uppercase">
  <xsl:call-template name="gentext">
    <xsl:with-param name="key">index.uppercase</xsl:with-param>
  </xsl:call-template>
</xsl:variable>

...
translate(substring(primary,1,1), $index.lowercase, $index.uppercase)
```

This approach seems viable at the first glance. Unfortunately it does not work because Muenchian method used for grouping depends on a key. Localized definition of the key is

```
<xsl:key name="letter" match="indexterm"
         use="translate(substring(primary,1,1),
                        $index.lowercase, $index.uppercase)"/>
```

But any decent XSLT processor should signal error at this instruction as XSLT recommendation forbids usage of variables inside key definitions.

So if we stay within limits of XSLT 1.0 we can not made indexing locale aware. The only solution is to edit lowercase and uppercase string constants in the `autoidx.xsl` file manually for usage with other than English language. This is quite easy as these strings are defined just once as an internal text entities and then used on many places. But this solution does not scale well so another solution was needed.

5. Internationalized index

The demand for internationalized indexing was rising and one day I was in the need for perfect Czech index. I was thinking about several approaches but the most viable was based on usage of user defined functions. Such function can label each word with ordinal number that identifies group in index and also position of this group inside index. This function can use external localization files to handle various languages in a different way. Value of this function can be used inside `xsl:key` definition and thus we can use Muenchian grouping method.

The current implementation of this function is designed in a way that it can correctly handle characters with diacritics and letters that are composed of two characters. The later is needed because some languages treat “ch” as a single letter which should sort between “c” and “d” in traditional Spanish or between “h” and “i” in Czech.

All information necessary for a correct grouping of index terms and collating these groups is stored together with other localization texts in localization files. Special structure is used as shown on Example 1, “Index localization data for Czech language”.

XSL•FO
RenderX

Example 1. Index localization data for Czech language

```
<l:letters>
  <l:l i="-1" />
  <l:l i="0">Symboly</l:l>
  <l:l i="1">A</l:l>
  <l:l i="1">a</l:l>
  <l:l i="1">Á</l:l>
  <l:l i="1">á</l:l>
  <l:l i="2">B</l:l>
  <l:l i="2">b</l:l>
  <l:l i="3">C</l:l>
  <l:l i="3">c</l:l>
  <l:l i="4"> </l:l>
  <l:l i="4"> </l:l>
  <l:l i="5">D</l:l>
  <l:l i="5">d</l:l>
  <l:l i="5"> </l:l>
  <l:l i="5"> </l:l>
  <l:l i="7">E</l:l>
  <l:l i="7">e</l:l>
  <l:l i="7">É</l:l>
  <l:l i="7">é</l:l>
  <l:l i="7"> </l:l>
  <l:l i="7"> </l:l>
  <l:l i="7">Ë</l:l>
  <l:l i="7">ë</l:l>
  <l:l i="8">F</l:l>
  <l:l i="8">f</l:l>
  <l:l i="9">G</l:l>
  <l:l i="9">g</l:l>
  <l:l i="10">H</l:l>
  <l:l i="10">h</l:l>
  <l:l i="11">Ch</l:l>
  <l:l i="11">ch</l:l>
  <l:l i="11">cH</l:l>
  <l:l i="11">CH</l:l>
  <l:l i="12">I</l:l>
  <l:l i="12">i</l:l>
  <l:l i="12">Í</l:l>
  <l:l i="12">í</l:l>
  <l:l i="13">J</l:l>
  <l:l i="13">j</l:l>
  <l:l i="14">K</l:l>
  <l:l i="14">k</l:l>
  <l:l i="15">L</l:l>
  <l:l i="15">l</l:l>
  <l:l i="16">M</l:l>
  <l:l i="16">m</l:l>
  <l:l i="17">N</l:l>
  <l:l i="17">n</l:l>
  <l:l i="17"> </l:l>
  <l:l i="17"> </l:l>
```

XSL•FO RenderX

```
  <l:l i="19">O</l:l>
  <l:l i="19">o</l:l>
  <l:l i="19">Ó</l:l>
  <l:l i="19">ó</l:l>
  <l:l i="19">Ö</l:l>
  <l:l i="19">ö</l:l>
  <l:l i="20">P</l:l>
  <l:l i="20">p</l:l>
  <l:l i="21">Q</l:l>
  <l:l i="21">q</l:l>
  <l:l i="22">R</l:l>
  <l:l i="22">r</l:l>
  <l:l i="23"> </l:l>
  <l:l i="23"> </l:l>
  <l:l i="24">S</l:l>
  <l:l i="24">s</l:l>
  <l:l i="25">Š</l:l>
  <l:l i="25">š</l:l>
  <l:l i="26">T</l:l>
  <l:l i="26">t</l:l>
  <l:l i="26"> </l:l>
  <l:l i="26"> </l:l>
  <l:l i="28">U</l:l>
  <l:l i="28">u</l:l>
  <l:l i="28">Ú</l:l>
  <l:l i="28">ú</l:l>
  <l:l i="28"> </l:l>
  <l:l i="28"> </l:l>
  <l:l i="28">Ü</l:l>
  <l:l i="28">ü</l:l>
  <l:l i="29">V</l:l>
  <l:l i="29">v</l:l>
  <l:l i="30">W</l:l>
  <l:l i="30">w</l:l>
  <l:l i="31">X</l:l>
  <l:l i="31">x</l:l>
  <l:l i="32">Y</l:l>
  <l:l i="32">y</l:l>
  <l:l i="32">Ý</l:l>
  <l:l i="32">ý</l:l>
  <l:l i="33">Z</l:l>
  <l:l i="33">z</l:l>
  <l:l i="34">Ž</l:l>
  <l:l i="34">ž</l:l>
</l:letters>
```

As you can see there is a separate `l` element for each letter that can occur at the start of an index term. Attribute `i` assigns group number to this letter. Letters that should appear within the same index group thus have the same value in this attribute. For example index terms starting with either “A”, “a”, “Á” or “á” will be placed in the same group. The label of group that will appear in the index is taken from the first `l` element with the corresponding group number. This means that words starting with variants of letter “a” will be in the group labeled with “A”.

The localization table is also able to cope with two character letters. The following definition ensures that terms starting with “ch” will be in a separate index group after “h” and before “i”. The order is defined by value stored in the attribute `i`.

```
   <l:l i="10">H</l:l>
   <l:l i="10">h</l:l>
   <l:l i="11">Ch</l:l>
   <l:l i="11">ch</l:l>
   <l:l i="11">cH</l:l>
   <l:l i="11">CH</l:l>
   <l:l i="12">I</l:l>
   <l:l i="12">i</l:l>
```

One of the goals of the DocBook XSL stylesheets is to be as portable as possible. For that reason I decided to implement above described indexing function as an EXSLT [http://www.exslt.org] function instead of proprietary Java/C/Python/… extension for particular XSLT processor. The most used XSLT implementations for DocBook processing are probably Saxon, xsltproc and Xalan. All of these programs claim support for EXSLT user defined function. Such user defined function is very similar to named XSLT template. The biggest difference is the fact that user defined function can be called directly from XPath expression as any other XPath function.

XSL•FO RenderX

Example 2. Function that returns index group number for a given term

```
<func:function name="i:group-index">
  <xsl:param name="term"/>

  <xsl:variable name="letters-rtf">
    <xsl:variable name="lang">
      <xsl:call-template name="l10n.language"/>
    </xsl:variable>

    <xsl:variable name="local.l10n.letters"
      select="($local.l10n.xml//l:i18n/l:l10n[@language=$lang]/
                l:letters)[1]"/>

    <xsl:variable name="l10n.letters"
      select="($l10n.xml/l:i18n/l:l10n[@language=$lang]/
                l:letters)[1]"/>

    <xsl:choose>
      <xsl:when test="count($local.l10n.letters) &gt; 0">
        <xsl:copy-of select="$local.l10n.letters"/>
      </xsl:when>
      <xsl:when test="count($l10n.letters) &gt; 0">
        <xsl:copy-of select="$l10n.letters"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:message>
          <xsl:text>No "</xsl:text>
          <xsl:value-of select="$lang"/>
          <xsl:text>" localization of index grouping
                    letters exists</xsl:text>
          <xsl:choose>
            <xsl:when test="$lang = 'en'">
              <xsl:text>.</xsl:text>
            </xsl:when>
            <xsl:otherwise>
              <xsl:text>; using "en".</xsl:text>
            </xsl:otherwise>
          </xsl:choose>
        </xsl:message>

        <xsl:copy-of select="($l10n.xml/l:i18n/l:l10n[@language='en']/
                              l:letters)[1]"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:variable>

  <xsl:variable name="letters"
              select="exslt:node-set($letters-rtf)/*"/>

  <xsl:variable name="long-letter-index"
              select="$letters/l:l[. = substring($term,1,2)]/@i"/>
  <xsl:variable name="short-letter-index"
```

```
                select="$letters/l:l[. = substring($term,1,1)]/@i"/>
  <xsl:variable name="letter-index">
    <xsl:choose>
      <xsl:when test="$long-letter-index">
        <xsl:value-of select="$long-letter-index"/>
      </xsl:when>
      <xsl:when test="$short-letter-index">
        <xsl:value-of select="$short-letter-index"/>
      </xsl:when>
      <xsl:otherwise>0</xsl:otherwise>
    </xsl:choose>
  </xsl:variable>
  <func:result select="number($letter-index)"/>
</func:function>
```

The largest part of function `i:group-index()` is code that loads the localization data from the correct place. Then function tries to find match for the first two characters of term (e.g. handling of “ch”). If there is no two letter match, then one letter match is used. Function returns number of the corresponding index group. If no letter match is found then function returns zero that represents group for symbols and other terms not starting with letter.

Using this function we can define supplementary key for grouping.

```
<xsl:key name="group-code"
         match="indexterm"
         use="i:group-index(primary)"/>
```

Now it is quite easy to modify original code for grouping index entries. We use our user defined function to form groups based on their number taken from localization file. The sort order of groups is also defined by this number.

```
<xsl:apply-templates
    select="//indexterm[count(.|key('group-code',
                                    i:group-index(primary))[1]) = 1]"
    mode="index-div">
  <xsl:sort select="i:group-index(primary)" data-type="number"/>
</xsl:apply-templates>
```

We must also slightly modify the code for handling an index group. This code emits group label and process each term belonging to this group just once.

```
<xsl:template match="indexterm" mode="index-div">
  <!-- Get the group index -->
  <xsl:variable name="key" select="i:group-index(primary)"/>

  <!-- Get the current language -->
  <xsl:variable name="lang">
    <xsl:call-template name="l10n.language"/>
  </xsl:variable>

  <!-- Output label for current index group -->
  <xsl:value-of select="i:group-letter($key)"/>

  <xsl:apply-templates select="key('group-code', $key)
```

```
                               [count(.|key('primary', primary)[1])=1]"
                       mode="index-primary">
    <xsl:sort select="primary" lang="{$lang}"/>
  </xsl:apply-templates>
</xsl:template>
```

We are using supplementary function `i:group-letter()` to return label of group with particular number.

The code that we presented up to this point was able to properly group index terms and order these groups according to language specific rules. But one problem still left unresolved—sorting of terms inside each index group. I decided to left this task to XSLT processors as they can use underlying implementation provided by virtual machine or operating system to do proper collating. Other processors allow you to specify your own collating sequence. For example if you want correct sort order for Czech in Saxon[4] you must create simple Java class, compile it and then add it into CLASSPATH. Other processor can provide similar functionality.

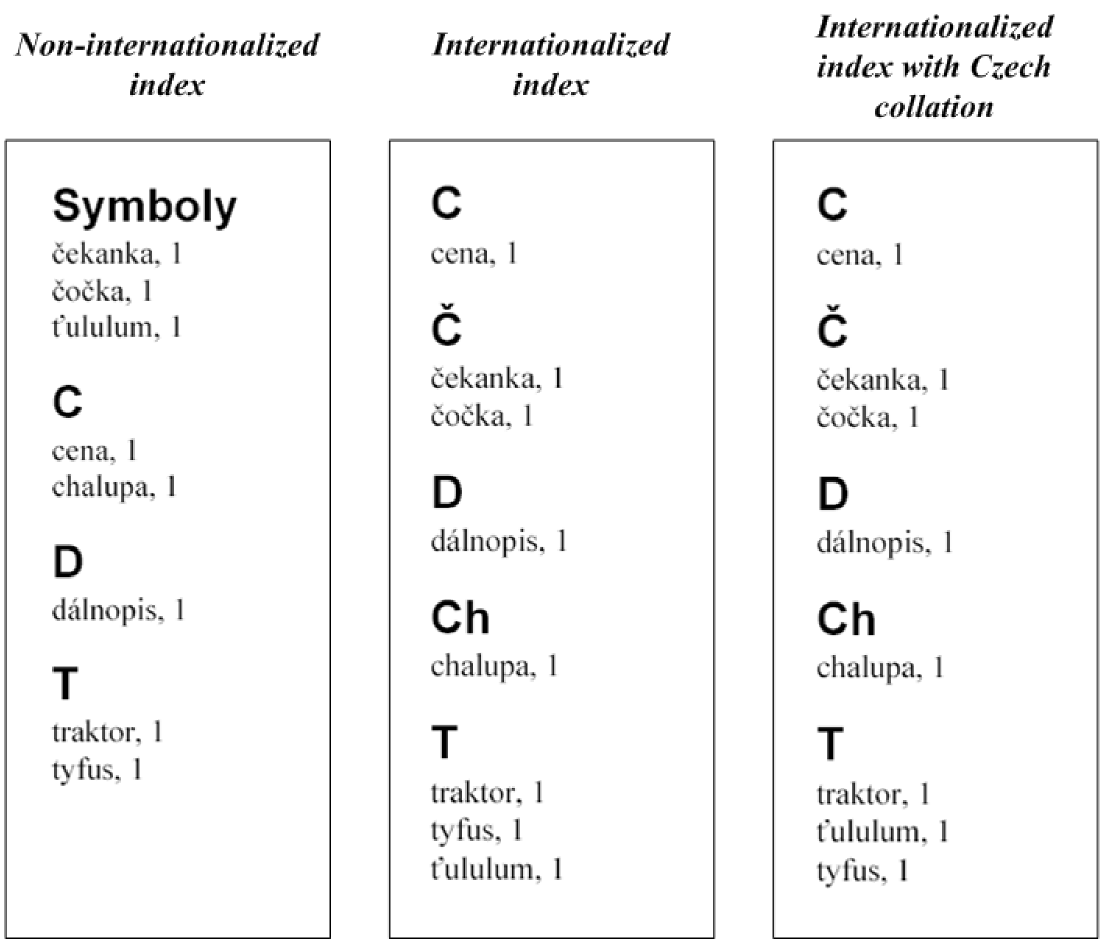
Example 3. Sample class that adds proper Czech collating support into Saxon

```
package com.icl.saxon.sort;

import java.text.Collator;
import java.util.Locale;

public class Compare_cs extends TextComparer
{

    int caseOrder = UPPERCASE_FIRST;

    public int compare(Object a, Object b)
    {

        Collator csCollator = Collator.getInstance(
                                 new Locale("cs", "cz"));
        return csCollator.compare(a, b);
    }

    public Comparer setCaseOrder(int caseOrder)
    {
        this.caseOrder = caseOrder;
        return this;
    }

}
```

Code presented in this paper shows only the most important parts and it is simplified little bit. If you want to study complete code you can look at files `common/autoidx-ng.xsl`, `html/autoidx-ng.xsl` and `fo/autoidx-ng.xsl` in the stylesheets distribution [http://docbook.sourceforge.net].

[4]Saxon version 6.5.x is used with the DocBook XSL stylesheets. Newer versions of Saxon can use JVM collation automatically without need for user defined classes.

XSL•FO RenderX



Non-internationalized index

Symboly
čekanka, 1
čočka, 1
ťululum, 1

C
cena, 1
chalupa, 1

D
dálnopis, 1

T
traktor, 1
tyfus, 1

Internationalized index

C
cena, 1

Č
čekanka, 1
čočka, 1

D
dálnopis, 1

Ch
chalupa, 1

T
traktor, 1
tyfus, 1
ťululum, 1

Internationalized index with Czech collation

C
cena, 1

Č
čekanka, 1
čočka, 1

D
dálnopis, 1

Ch
chalupa, 1

T
traktor, 1
ťululum, 1
tyfus, 1

Figure 1. Sample index processed with different configurations

6. Integration with DocBook stylesheets and compatibility

EXSLT is not supported in all XSLT implementations and thus we can not add internationalized indexing into the stock stylesheets. This would break compatibility for people who are not interested in internationalized indexes. Different deployment method was thus selected. Internationalized stylesheets are part of distribution but are not included by default.

If we want to use internationalized indexing features of the stylesheets we must create a customization layer that overrides default index generating templates by including a small `autoidx-ng.xsl` stylesheet.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">

<xsl:import
  href="http://docbook.sf.net/release/xsl/current/fo/docbook.xsl"/>
<xsl:include
  href="http://docbook.sf.net/release/xsl/current/fo/autoidx-ng.xsl"/>
```

XSL•FO
RenderX

```

<!-- Parameter settings and other modifications of stylesheet -->

</xsl:stylesheet>
```

This customization layer is then used instead of stock stylesheets.

Internationalized indexing method is known to work with Saxon 6.5.3 and Xalan 2.6.0. I put substantial amount of time into porting this code to xsltproc but I was unsuccessful up to this time. Authors of xsltproc interpret EXSLT specification in a very restrictive point of view of XSLT 1.0 and does not allow usage of variables inside user defined functions that are used in keys. For that reason I created alternative implementation of internationalized indexing that did not use Muenchian grouping method that depends on `xsl:key`. But grouping is very slow without keys—xsltproc is 40 times slower than Saxon that can use keys for this task. Moreover xsltproc has very poor support for specifying user defined collation sequences so I give up xsltproc support until these problems are resolved.

7. Print output issues

Generating a printed back-of-the-book index in XSL is a two phase process. The first phase is a XSLT transformation that converts a source DocBook document into a set of abstract formatting objects. Page numbers for the index entries are not known at this moment. The actual rendering and page number evaluation takes part during the second formatting phase, which is performed by a FO processor like FOP, XEP or XSL Formatter. The problem arises when one index term occurs twice within a page. In this case, the index contains duplicate page numbers for this entry. This serious drawback can be overcome in two ways. The first solution utilizes the FO processor, that implements a vendor extension for the index generation. The other possibility is to use multiple passes over a document to detect and remove the duplicities.

The vendor extensions are supported in the two most known commercial FO processors—XEP [http://www.renderx.com/] and XSL Formatter [http://www.antennahouse.com/]. The DocBook XSL stylesheets contain support for these FO implementations and are able to add special indexing elements and attributes into FO output. For each FO processor there is a parameter turning on special indexing features. For instance XEP should be invoked by the following command line

```
xep -xml document.xml -xsl .../fo/docbook.xsl -param xep.extensions=1
```

In the real world we usually change behaviour of the stylesheets by customizing more then one parameter. The best practice is then to create a customization layer, that imports stock stylesheets and sets all necessary parameters.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">

<xsl:import
  href="http://docbook.sf.net/release/xsl/current/fo/docbook.xsl"/>

<xsl:param name="paper.type" select="'A4'"/>
<xsl:param name="xep.extensions" select="1"/>

</xsl:stylesheet>
```

If you prefer XSL Formatter over XEP, you can use a similar parameter `axf.extensions` to turn on the XSL Formatter support. Use of these parameters results in removing duplicate page numbers and in creating a page range for continuous sequences of page numbers. For example if a single index entry occurs on the following pages:

XSL•FO RenderX

```
5, 5, 8, 9, 10, 37
```

the output will be more reasonable and esthetic in the following way:

```
5, 8–10, 37
```

When we are using FO processor that does not support indexing extension, we must employ more difficult procedure. This is also the case of the open-source FOP processor [http://xml.apache.org/fop/]. We must process the document twice. The first pass is done with the `make.index.markup` parameter set. The resulting PDF will contain a XML markup for index entries and page numbers. This PDF can be converted to a plain text from which the XML markup is extracted. The duplicates are then removed and the modified XML fragment of the index is now used to get the proper PDF. This process is a real hackery and it does not work very well for languages that use characters outside the ISO Latin 1 as the FOP does not insert the proper Unicode mapping vector for embeded fonts. This technique was invited by G. Ken Holman [http://www.cranesoftwrights.com/resources/bbi/index.htm].

8. World after XSLT 2.0 and XSL-FO 1.1

We have seen that it is probably impossible to implement internationalized indexing using only XSLT 1.0 features. Our implementation used two EXSLT extensions—user defined functions and `node-set()` function. Fortunately authors of upcoming version XSLT 2.0 listen very carefully to user needs and XSLT 2.0 offers many new features that can significantly simplify our indexing task.

New `xsl:function` instruction offers a standard way for declaring user defined functions. Such function can be used in any XPath expression. There is no need for using Muenchian grouping method as XSLT 2.0 offers new instruction `xsl:for-each-group`. Alike there is no need for using `node-set()` function because XSLT 2.0 throws away result tree fragments.

Creating an index in XSLT 2.0 is pretty straightforward using new grouping facilities.

```
<!-- Get the current language -->
<xsl:variable name="lang">
  <xsl:call-template name="l10n.language"/>
</xsl:variable>

<!-- Create index groups -->
<xsl:for-each-group select="//indexterm"
                    group-by="i:group-index(primary)">
  <xsl:sort select="i:group-index(primary)"/>

  <!-- Output label for current index group -->
  <xsl:value-of select="i:group-letter(current-grouping-key())"/>

  <!-- Group index terms in one group -->
  <xsl:for-each-group select="current-group()" group-by="primary">
    <xsl:sort select="primary" lang="{$lang}"/>
    <xsl:apply-templates select="." mode="index-primary">
  <xsl:for-each-group>
</xsl:for-each-group>
```

XSLT 2.0 also offers better interface for specifying user defined collations which is important for proper sorting of terms inside an index group. Saxon 7/8, only reasonable XSLT 2.0 WD implementation available, supports all collations provided by underlying Java VM.

XSL•FO RenderX

To summarize it: XSLT 2.0 offers great new features that made generating of internationalized indexes very easy compared to XSLT 1.0 + EXSLT solution. The new solution does not need to use any extensions and it is thus more portable. The only drawback of XSLT 2.0 is its slow standardization process. For the last four years the answer to question “When the XSLT 2.0 will be finished?” was “Probably in the next year.” I hope this year is the last year when this answer is correct. It can be expected that after finalizing specification more implementations will be placed on the marked. But if you are early adopter you can use XSLT 2.0 with Saxon 8 right now. Saxon 8 is mature enough for production use.

Development of new version of XSL-FO is not in as advanced stage as XSLT 2.0 but there is Working Draft for XSL-FO 1.1. It introduces new formatting objects for dealing with index and suppressing duplicate page numbers in index. These new constructs are replacement for vendor extensions or multi pass processing that must be done these days.

9. Further work

The current implementation of internationalized indexing is known to work in Saxon and in Xalan. Index localization data are available for seven languages: Czech, Danish, German, English, Spanish, French and Turkish. All of these files were created manually except English one. English one was generated from Unicode character database and contain all accented variants of 26 letters. These variants are always placed in the same index group as unaccented one.

Currently we are planning to add support for indexing CJKV languages. This is especially challenging task as there are several thousands of such characters and my knowledge of CJKV languages is equal to zero. But with the help of other resources things becoming clearer. One thing that is obvious right now is need for different layout of index localization files. Requirements for CJKV languages are very different. Index terms are grouped by number of strokes or by radicals in glyph. Group label is not just first letter of an index term as in Latin based languages. We will probably use alternative layout of index localization data more appropriate for CJKV languages and the stylesheet will be adapted to handle data in both formats.

In the long term the stylesheets will migrate to XSLT 2.0 where the internationalized indexing will be on by default. However this new implementation probably would not start before XSLT 2.0 reaches at least Candidate Recommendation status.

Meanwhile work on compatibility with another XSLT implementations is expected as well as adding support of new languages. If your preferred language is currently not supported in internationalized indexing we would appreciate if you can provide index localization data to the DocBook XSL stylesheets project.

10. Related work

Original indexing code in the DocBook XSL stylesheets comes from Jeni Tennison. It was later modified to support several new features but overall design of code remained intact.

After I finished first prototype of internationalized indexing I come across Kimber's and Reynold's paper [FOIDX] that deals with the same problem. Their solution is implemented as Java extension for Saxon and it is ready to support CJK languages. Advantage of solution used in the DocBook XSL stylesheet is better compatibility—at least two processors (Saxon and Xalan) are able to do internationalized indexing.

11. Conclusion

In the article I shown general method for creating internationalized back-of-the-book indexes using XSLT 1.0 with EXSLT extensions. Advantage of this method is that it works in more than one XSLT implementation. Integration of this method into the DocBook XSL stylesheets was presented. Internationalized indexing is not turned on by default

XSL•FO
RenderX

to maintain compatibility with XSLT processors that do not support EXSLT. New emerging standards like XSLT 2.0 and XSL-FO 1.1 will make internationalized index generation easier and will allow make it default feature.

Bibliography

[XSLFO] *Extensible Stylesheet Language (XSL) Version 1.0*, Sharon Adler, Anders Berglund, Jeff Caruso, Stephen Deach, Paul Grosso, Eduardo Gutentag, Alex Milowski, Scott Pernell, Jeremy Richman, Steve Zilles, W3C, 2001. Available at http://www.w3.org/TR/xsl.

[XSLFO11] *Extensible Stylesheet Language (XSL) Version 1.1* Anders Berglund, W3C, 2003. Available at http://www.w3.org/TR/xsl11/.

[XSLT1] *XSL Transformations (XSLT) Version 1.0*, James Clark, W3C, 1999. Available at http://www.w3.org/TR/xslt.

[XSLT2] *XSL Transformations (XSLT) Version 2.0*, Michael Kay, W3C, 2003. Available at http://www.w3.org/TR/xslt20/.

[DBIDX] *Mastering DocBook Indexes*, Jirka Kosek, xml.com, July 14, 2004. Available at http://www.xml.com/pub/a/2004/07/14/dbndx.html.

[FOIDX] *Internationalized Back-of-the-Book Indexes for XSL Formatting Objects*, Eliot W. Kimber and Joshua Reynolds, Extreme Markup Languages 2002 Paper, August 6–9, 2002. Available at http://www.mulberrytech.com/Extreme/Proceedings/html/2002/Kimber01/EML2002Kimber01-toc.html.

[DBXSL] *DocBook XSL: The Complete Guide*, Robert Stayton, Sagehill Enterprises, 2003, ISBN 0-9741521-1-0. Available at http://www.sagehill.net/docbookxsl/index.html.

[TDG] *DocBook – The Definitive Guide*, Norman Walsh, Leonard Muellner, O'Reilly, 1999, ISBN 156592-580-7. Available at http://www.docbook.org/tdg/en/html/docbook.html.

Biography

Jirka **Kosek**
University of Economics
Prague
Czech Republic
jirka@kosek.cz

Jirka Kosek [http://www.kosek.cz] is the freelance writer, consultant, trainer, university teacher, open source developer and PhD student. He has written several books about XML, HTML and PHP. He is a member of DocBook developers team, which is developing and improving DSSSL and XSL stylesheets for formatting DocBook documents. He is also a member of OASIS DocBook Technical Committee.

XSL•FO
RenderX