Zpravodaj Československého sdružení uživatelů TeXu

Zdeněk Wagner

Zpracování pomocných TeXových souborů pomocí XSLT 2.0

Zpravodaj Československého sdružení uživatelů TeXu, Vol. 16 (2006), No. 1, 40–53

Persistent URL: http://dml.cz/dmlcz/150007

Summary: TrueType fonts, TEX and Czech language

TrueType and OpenType fonts have several advantages compared to the well-known Type 1 fonts, which have been used for years by TEX users. In this article a way how to use TrueType fonts with TEX for typesetting in Czech and Slovak language is presented. While instructions are specific for encoding XL2 and XT2 (which are compatible with ISO 8859-2), the encoding vectors can be easily modified to support other encodings, like Cork.

Ondřej Jakubčík
e-mail: ojakubcik@seznam.cz

Zpracování pomocných TEXových souborů pomocí XSLT 2.0

Zdeněk Wagner

Článek nastiňuje možnosti zpracování pomocných TEXových souborů procesorem XSLT 2.0. Koncept je demonstrován reimplementací programu MakeIndex. Autor se též zamýšlí nad možností reimplementace BibTEXu pouze pomocí XSLT.

1. Úvod

V moderních operačních systémech se začíná prosazovat jako standard kódování Unicode, resp. UTF-8. V programech se tak projevuje obdoba problému Y2K. Zatímco dosavadní kódování vystačila pro téměř každý jazyk s osmibitovými znaky, nyní potřebujeme šestnáctibitové. Úprava programů však není zcela triviální. Kódování UTF-8 umožňuje úsporu místa na disku tím, že znaky US abecedy zabírají pouze jeden bajt, zatímco akcentované znaky a znaky východoasijských jazyků jsou vícebajtové. Navíc může být program provázán s dalšími pomocnými soubory, např. s fonty. Programátor tak musí řešit komplexní problém.

Stejná potíž postihla též TEX a jeho podpůrné programy. Zpracování vstupu v UTF-8 lze v TEXu řešit různými metodami, z nichž nejlepší je encTEX [1], neboť z hlediska programátora maker se znak stále chová jako znak. Makro nikdy nevidí polovinu znaku. Některé úlohy se však řeší pomocí externích programů a některé z nich ještě na zpracování UTF-8 upraveny nejsou.

UTF-8 musí být podporováno ve všech programech, jež splňují standardy XML. Nabízí se tedy možnost, že pomocné soubory budou překonvertovány do XML, zpracovýny nějakým vhodným nástrojem a opět převedeny do formátu, který TEX umí načíst. Tuto úlohu nám usnadňuje poměrně nový standard XSLT 2.0 [2], který obsahuje funkci pro zpracování obyčejných textových souborů. Tento jazyk implementuje XPath 2.0 [3]. Ukázku takového zpracování předvedl Michael Kay na konferenci XML Prague 2005 [4].

2. Reimplementace programu MakeIndex pomocí XSLT 2.0

Cílem této ukázky není vytvoření programu s naprosto stejnou funkcí. Chyby vstupu mohou být vyřešeny jiným způsobem, ale na druhé straně bude zde vytvořený program obohacen o několik nových vlastností. Systém byl testován s procesorem XSLT Saxon-B 8.5 [5] v operačním systému OS/2 Warp 4 s Javou 1.4.1 od GoldenCode [6] a 1.4.2_05 od firmy Innotek [7]. Zde popisované transformační styly jsou volně dostupné [8]. Nebudeme proto uvádět kompletní kód. Budeme rekonstruovat postup jejich vývoje a vysvětlíme si vybrané zajímavosti.

V ukázkách budeme používat prefixy několika jmenných prostorů. Především `xsl` je jmenným prostorem XSLT, jenž je svázán s URI `http://www.w3.org/1999/XSL/Transform`. Prefixem `xs` označíme jmenný prostor XML Schema, jehož URI je `http://www.w3.org/2001/XMLSchema`. Nakonec budeme vytvářet vlastní funkci, která musí mít neprázdné URI jmenného prostoru. URI může být zcela libovolné. Zde použijeme `http://icebearsoft.euweb.cz` a svážeme je s prefixem `zw`.

2.1. Načtení TEXového souboru

Po prvním zpracování vstupu LATEXem vznikne soubor s příponou `idx`. Tento pomocný soubor je vstupem pro MakeIndex. My jej ovšem budeme načítat procesorem XSLT. K tomu je určena funkce `unparsed-text()`. Jejím prvním parametrem je URI souboru, jejž chceme načíst. Předpokládá se, že soubor je uložen v kódování UTF-8. Má-li soubor jiné kódování, zadáme jej v druhém parametru. Chceme uživatelům umožnit, aby si kódování mohli zadat sami, přičemž UTF-8 ponecháme jako default. V transformačním stylu `parse-index.xsl` proto nadefinujeme parametr:

```
<xsl:param name='enc' as='xs:NMTOKEN'>utf-8</xsl:param>
```

Celý soubor pak lze vložit do jedné řetězcové proměnné příkazem:

```
<xsl:variable name='idx-content' as='xs:string'
              select='unparsed-text($index, $enc)'/>
```

přičemž `$index` je jméno požadovaného souboru. My to však uděláme trošku jinak. MakeIndex je schopen vzít vstup z několika souborů. Tuto vlastnost lze implementovat velmi snadno. Nadefinujeme si parametry transformačního stylu:

```
<xsl:param name='index' as='xs:string' required='yes'/>
<xsl:param name='file-separator' as='xs:string',</xsl:param>
```

Jména souborů, zadaná ve tvaru `index=seznam`, nyní rozdělíme použitím funkce

```
tokenize($index, $file-separator)
```

Získáme tím posloupnost tokenů, z nichž každý reprezentuje URI jednoho souboru. Načteme je tedy v cyklu:

```
for $fn in (tokenize($index, $file-separator))
    return unparsed-text($fn, $enc)
```

Na závěr vše slepíme dohromady funkcí `string-join()`. Obsah všech vstupních souborů v jednom řetězci však není to, co by se nám pro další zpracování hodilo. Víme, že jednotlivé položky jsou zapsány pomocí maker, obvykle `\indexentry`. Využijeme tedy další funkce, již nám XSLT 2.0 nabízí, a to zpracování řetězce pomocí regulárních výrazů v perlovské syntaxi. Každé makro s parametry tedy převedeme na element uložený pouze v paměti procesoru XSLT. Kostra příkazu vypadá takto:

```
<xsl:variable name='index-items' as='item()*'>
  <xsl:analyze-string
       select='string-join(for $fn in
               (tokenize($index, $file-separator))
               return unparsed-text($fn, $enc), "")'
     regex='\\(\S+?)\s*\{{(.+?)\}}\{{(-?\d+?)\}}\r?\n' flags='s'>
    <xsl:matching-substring>
      <xsl:variable name='entry' as='xs:string'
           select='regex-group(1)'/>
      <xsl:variable name='page' as='xs:integer'
           select='xs:integer(regex-group(3))'/>
      ... <!-- Kód pro analýzu argumentů -->
    </xsl:matching-substring>
    <xsl:non-matching-substring>
      <xsl:message>
The input file contains illegal entry:
<xsl:value-of select='normalize-space(.)'/>
      </xsl:message>
    </xsl:non-matching-substring>
  </xsl:analyze-string>
</xsl:variable>
```

Položky musí vyhovovat zadanému regulárnímu výrazu. Nevyhovující řetězce zapomeneme a na terminál zobrazíme chybovou zprávu.

Všimněte si, že se vůbec nestaráme o jméno makra, v němž je položka uložena, pouze si toto jméno uložíme. Uschováme si též číslo stránky. Vlastní položka může mít komplikovanější syntaxi. Text proto znovu rozebereme pomocí regulárních výrazů. MakeIndex umožňuje definovat různé druhy oddělovačů ve stylovém souboru. My je nadefinujeme pomocí parametrů transformačního stylu, jimž ponecháme analogická jména a totožné defaultní hodnoty. Některé parametry budou využívány uvnitř regulárních výrazů jako *attribute value template*. Speciální znaky proto musí být uvozeny zpětným lomítkem:

```
<xsl:param name='encap' as='xs:string'>\|</xsl:param>
<xsl:param name='actual' as='xs:string'>@</xsl:param>
<xsl:param name='escape' as='xs:string'>\\</xsl:param>
<xsl:param name='level' as='xs:string'>!</xsl:param>
<xsl:param name='quote' as='xs:string'>"</xsl:param>
```

Další znaky budou užity jen v textových řetězcích, takže zpětné lomítko se použít nesmí:

```
<xsl:param name='range-open' as='xs:string'>(</xsl:param>
<xsl:param name='range-close' as='xs:string'>)</xsl:param>
```

Vlastní text položky zpracujeme tímto kódem:

```
<xsl:analyze-string select='normalize-space(regex-group(2))'
     regex='(.+[^{$escape}{$quote}]){$encap}\s*(.+)'>
  <xsl:matching-substring>
    <xsl:call-template name='make-index-item'>
      <xsl:with-param name='entry' tunnel='yes' select='$entry'/>
      <xsl:with-param name='page' tunnel='yes' select='$page'/>
      <xsl:with-param name='style' tunnel='yes'
           select='regex-group(2)'/>
      <xsl:with-param name='text' select='regex-group(1)'/>
    </xsl:call-template>
  </xsl:matching-substring>
  <xsl:non-matching-substring>
    <xsl:call-template name='make-index-item'>
      <xsl:with-param name='entry' tunnel='yes' select='$entry'/>
      <xsl:with-param name='page' tunnel='yes' select='$page'/>
      <xsl:with-param name='text' select='.'/>
    </xsl:call-template>
  </xsl:non-matching-substring>
</xsl:analyze-string>
```

V kódu normalizujeme mezery, takže `brouk␣␣␣Pytlík` bude zkomprimován do formy `brouk␣Pytlík`. Odstraní se i případné mezery na začátku a konci řetězce a řádkové zlomy se též nahradí mezerou. Regulární výraz odpovídá po expanzi proměnných (v perlovském zápisu) `/(.+[^\\"])\|\s*(.+)/`. Tímto výrazem oddělíme text položky od označení stylu zobrazení stránkové číslice. Při předávání parametrů šabloně `make-index-item` nastavíme stribut `tunnel='yes'`. Šablonu totiž budeme volat rekurzivně. Její kostra vypadá takto:

```
<xsl:template name='make-index-item'>
  <xsl:param name='entry' as='xs:string' required='yes'
                          tunnel='yes'/>
  <xsl:param name='page' as='xs:integer' required='yes'
                         tunnel='yes'/>
  <xsl:param name='style' as='xs:string' tunnel='yes'
       select='$default-page-style'/>
  <xsl:param name='text' as='xs:string' required='yes'/>
  <xsl:element name='{$entry}'>
    <xsl:analyze-string select='$text'
         regex='(.+?[^{$escape}{$quote}]){$level}\s*(.+)'>
    ...
    </xsl:analyze-string>
  </xsl:element>
</xsl:template>
```

Účelem této šablony a šablon z ní volaných je oddělení víceúrovňových rejstříkových položek. Proto zpracujeme text rekurzivně, přičemž na rozdíl od programu MakeIndex není zde hloubka vnoření omezena. Atribut `tunnel='yes'` způsobí, že parametry se stejnou hodnotou „protunelují“ do rekurzivně volané šablony, aniž bychom je museli v elementu `<xsl:call-template>` explicitně uvádět. Studium kódu ponecháme čtenářům za domácí úkol.

Nyní nastal čas k tomu, abychom ověřili funkčnost vytvořeného stylu. Text byl přetransformován do sekvence elementů, jejichž jména odpovídají jménům maker ve vstupních souborech. Vše si můžeme vypsat do XML souboru šablonou:

```
<xsl:template name='parsed-index'>
  <parsed-index>
    <xsl:copy-of select='$index-items'/>
  </parsed-index>
</xsl:template>
```

Aby to nebylo tak jednoduché, předpokládejme, že soubory nazvané `file1.idx` a `file2.idx` pro nás někdo vytvořil v Linuxu v kódování ISO-8859-2, je nutno je zpracovat současně, ale výsledný `file.xml` si chceme prohlédnout v OS/2

v kódování CP852. Zařídíme to následujícím příkazem (celý text musíte zapsat na jeden řádek):

```
saxon8 -o file.xml -it parsed-index parse-index.xsl
  index=file1.idx,file2.idx enc=iso-8859-2 !encoding=cp852
```

2.2. Abecední řazení pomocného souboru

Řadicí algoritmus je implementován v samostatném souboru `sort-inxex.xsl`. Důvod takové modularizace si vysvětlíme později. Nevýhodou je, že tento styl již nelze použít samostatně, ale musíme oba soubory importovat do jednoho stylu např. takto:

```
<?xml version='1.0' encoding='utf-8'?>
<xsl:stylesheet version='2.0'
     xmlns:xsl='http://www.w3.org/1999/XSL/Transform'
     xmlns:xs='http://www.w3.org/2001/XMLSchema'
     exclude-result-prefixes='#all'>
<xsl:import href='parse-index.xsl'/>
<xsl:import href='sort-index.xsl'/>
</xsl:stylesheet>
```

Uživateli dáme šanci, aby si zvolil jazyk, podle jehož pravidel se má rejstřík řadit. Použijeme k tomu parametry stylu s vhodnou defaultní hodnotou:

```
<xsl:param name='case-order'
           as='xs:NMTOKEN'>lower-first</xsl:param>
<xsl:param name='lang' as='xs:NMTOKEN'>en</xsl:param>
```

Nadefinujeme si též parametr pro zadání jména elementu, jejichž abecedně seřazený seznam se má na výstupu objevit:

```
<xsl:param name='keyword' as='xs:NMTOKEN'>indexentry</xsl:param>
```

Podobně jako v minulém případě si budeme chtít výstup otestovat. Bude nám k tomu sloužit šablona pojmenovaná `sorted-index`. Předpokládáme, že ve finální verzi bude tento styl spolupracovat s dalšími styly a možná bude užitečné, abychom jinými prostředky dokázali ovlivnit metodu řazení. Testovací šablonu tedy definujeme s využitím tunelových parametrů:

```
<xsl:template name='sorted-index'>
  <xsl:param name='entry' as='xs:string' select='$keyword'/>
  <xsl:param name='x-lang' as='xs:NMTOKEN' select='$lang'/>
  <xsl:param name='x-case-order' as='xs:NMTOKEN'
             select='$case-order'/>
```

```
  <xsl:param name='items' as='item()*' select='$index-items'/>
  <sorted-index>
    <xsl:call-template name='sort-index-items'>
      <xsl:with-param name='entry' tunnel='yes' select='$entry'/>
      <xsl:with-param name='x-lang' tunnel='yes'
           select='$x-lang'/>
      <xsl:with-param name='x-case-order' tunnel='yes'
           select='$x-case-order'/>
      <xsl:with-param name='items' as='item()*' select='$items'/>
    </xsl:call-template>
  </sorted-index>
</xsl:template>
```

Samostudiem jste si mohli zjistit, že elementy, jejichž sekvence je uložena v proměnné `$index-items`, obsahují vnořený element `<key>` s řadicím klíčem, element `<text>` s vlastním textem položky (může se shodovat s klíčem a buď vnořenou položku, nebo element `<page>` s číslem stránky. Řazení tedy provedeme víceúrovňově touto rekurzivní šablonou:

```
<xsl:template name='sort-index-items'>
  <xsl:param name='entry' as='xs:string' required='yes'
             tunnel='yes'/>
  <xsl:param name='x-lang' as='xs:string' required='yes'
             tunnel='yes'/>
  <xsl:param name='x-case-order' as='xs:string' required='yes'
             tunnel='yes'/>
  <xsl:param name='items' as='item()*' select='$index-items'/>
  <xsl:for-each-group select='$items[name() = $entry]'
                      group-by='key'>
    <xsl:sort select='current-grouping-key()' data-type='text'
              case-order='{$x-case-order}' lang='{$x-lang}'/>
    <xsl:variable name='key' as='xs:string'
                  select='current-grouping-key()'/>
    <xsl:for-each-group select='current-group()' group-by='text'>
      <xsl:sort select='current-grouping-key()' data-type='text'
                case-order='{$x-case-order}' lang='{$x-lang}'/>
      <xsl:element name='{$entry}'>
        <xsl:attribute name='id'>
          <xsl:value-of select='$key'/>
        </xsl:attribute>
        <xsl:attribute name='text'>
          <xsl:value-of select='current-grouping-key()'/>
        </xsl:attribute>
```

```
        <xsl:call-template name='sort-index-items'>
          <xsl:with-param name='items'
               select='current-group()/element()[name()=$entry]'/>
        </xsl:call-template>
        <xsl:call-template name='process-page-numbers'>
          <xsl:with-param name='items'
               select='current-group()/page'/>
          <xsl:with-param name='keytext'
               select='if ($key = current-grouping-key())
               then $key
               else concat($key,$actual,current-grouping-key())'/>
        </xsl:call-template>
      </xsl:element>
    </xsl:for-each-group>
  </xsl:for-each-group>
</xsl:template>
```

Vybrané elementy nejprve seřadíme a seskupíme podle klíče, potom podle textu položky. Následně rekurzivním voláním seřadíme vnořené elementy a nakonec zpracujeme čísla stran. Všimněte si, že jsme neuvedli tunelové parametry, neboť procesor XSLT to provede za nás.

Při zpracování čísel stran se musíme vypořádat jednak se styly, jednak s rozsahy stran. Nejprve expandujeme explicitně uvedené rozsahy a uložíme výsledek do proměnné `$expanded-ranges`. V šabloně `$process-page-numbers` k tomu slouží kód (vynechali jsme chybové zprávy):

```
<xsl:variable name='expanded-ranges' as='item()*'>
  <xsl:for-each-group select='$items[@range]'
       group-starting-with='*[@range = "open"]'>
    <xsl:variable name='style' as='xs:string'
         select='self::*[1]/@style'/>
    <xsl:for-each select='self::*[1]/@number
                  to current-group()[last()]/@number'>
      <page number='{.}' style='{$style}'/>
    </xsl:for-each>
  </xsl:for-each-group>
</xsl:variable>
```

Nyní seřadíme čísla stran, která nepatří do explicitně zadaných rozsahů, společně se sekvencí vzniklou v minulém kroku. Řadíme podle čísla strany a podle stylu zobrazení. Výsledek uložíme do proměnné `$sorted-pages`:

```
<xsl:variable name='sorted-pages' as='item()*'>
  <xsl:for-each select='$items[not(@range)], $expanded-ranges'>
```

```
    <xsl:sort select='@number' data-type='number'/>
    <xsl:sort select='@style'/>
    <xsl:copy-of select='.'/>
  </xsl:for-each>
</xsl:variable>
```

Nyní potřebujeme skupiny sousedních stran seskupit do rozsahů, přičemž se nemusí jednat jen o rozsahy zadané explicitně. Rozsahy musíme seskupovat samostatně pro jednotlivé styly. Opět se nám bude hodit cyklus pro skupiny:

```
<xsl:variable name='collapsed-ranges' as='item()*'>
  <xsl:for-each-group select='$sorted-pages'
                      group-adjacent='@style'>
    <xsl:variable name='style' as='xs:string'
                  select='self::*/@style'/>
    <xsl:variable name='distinct-pages' as='xs:integer*'
         select='distinct-values(current-group()/@number)'/>
    ...
  </xsl:for-each-group>
</xsl:variable>
```

Funkcí `distinct-values()` jsme vyloučili opakované prvky. Hodnota je poslední (nebo jedinou) stránkou v rozsahu, pokud následující hodnota je nejméně o 2 vyšší, nebo se jedná o poslední prvek v sekvenci. Takovým elementům přidáme atribut `range='close'`:

```
<xsl:variable name='pages' as='item()*'>
  <xsl:for-each select='1 to count($distinct-pages)'>
    <page number='{subsequence($distinct-pages, ., 1)}'
          style='{$style}'>
      <xsl:variable name='pg' as='xs:integer*'
           select='subsequence($distinct-pages, ., 2)'/>
      <xsl:if test='. = count($distinct-pages) or
                      min($pg) &lt; max($pg) - 1'>
        <xsl:attribute name='range'>close</xsl:attribute>
      </xsl:if>
    </page>
  </xsl:for-each>
</xsl:variable>
```

Rozsahy více než dvou stran nahradíme jediným elementem, v němž bude konec rozsahu vložen do atributu `rangeTo`. Atribut `range` ze všech elementů odstraníme.

```
<xsl:for-each-group select='$pages'
                    group-ending-with='*[@range="close"]'>
  <xsl:choose>
    <xsl:when test='current-group()[last()]/@number
              - self::*/@number &gt; 1'>
      <page number='{self::*/@number}'
            rangeTo='{current-group()[last()]/@number}'
            style='{self::*/@style}'/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:for-each select='current-group()'>
        <xsl:copy>
          <xsl:copy-of select='@* except @range'/>
        </xsl:copy>
      </xsl:for-each>
    </xsl:otherwise>
  </xsl:choose>
</xsl:for-each-group>
```

Výslednou sekvenci opět seřadíme, a to podle počátečních a koncových čísel rozsahů a podle stylů:

```
<xsl:for-each select='$collapsed-ranges'>
  <xsl:sort select='@number' data-type='number'/>
  <xsl:sort select='if (@rangeTo) then @rangeTo
                    else -99999' data-type='number'/>
  <xsl:sort select='@style'/>
  <xsl:copy-of select='.'/>
</xsl:for-each>
```

Takto napsaný kód si neporadí správně s případem, kdy do rejstříku vkládáme heslo uvedené na některé z úvodních stránek číslovaných římskými číslicemi. Takový přístup není známkou správného stylu psaní knih, takže obtíže spojené s implementací této vlastnosti nestojí za vynaložené úsilí.

2.3. Formátování výstupu

Formátování výstupu je naprogramováno ve stylu `format-index.xsl`. Opět jej nelze použít samostatně, ale jen společně s předchozími dvěma styly. Zde je specifikována výstupní metoda `text`, ale není uvedeno kódování, aby si jej uživatel mohl zadat sám. Atribut `encoding` elementu `<output/>` není *attribute value template*, takže jej nelze převzít z hodnoty parametru `$enc`.

V transformačním stylu nejprve do pomocné proměnné uložíme seřazený seznam:

```
<xsl:variable name='sorted-list' as='element()'>
  <xsl:call-template name='sorted-index'/>
</xsl:variable>
```

Pokud nechceme mít hlavičky s počátečními písmeny, vytvoříme výstup jednoduchou šablonou:

```
<xsl:template name='noheaders'>
  <xsl:param name='sorted-items' as='element()'
             select='$sorted-list'/>
  <xsl:text>\begin{theindex}</xsl:text>
  <xsl:apply-templates select='$sorted-items'/>
  <xsl:text>&#10;\end{theindex}&#10;</xsl:text>
</xsl:template>
```

Šablony pro formátování výstupu si nastudujte za domácí úkol. Opět využijeme rekurzivní volání, abychom zpracovali vnořené položky.

Hlavičky s počátečními symboly představují pouze drobnou komplikaci. K řešení nám poslouží cyklus přes skupiny, kde využijeme seskupení podle prvního znaku konvertovaného na velké písmeno. V češtině a slovenštině musíme ještě ošetřit CH. To však není vše. Zbývají akcentovaná písmena, zejména dlouhé samohlásky, jež nemají primární řadicí platnost. Vytvoříme si proto nejprve funkci, která určí hodnotu klíče při řazení podle českých nebo slovenských pravidel:

```
<xsl:function name='zw:key'>
  <xsl:param name='key' as='xs:string'/>
  <xsl:value-of
       select="if ($lang != 'cs' and $lang != 'sk') then $key
       else translate($key,
           'ÁÄĎÉĚÍĹĽŇÓÖŐŤÚŮÜŰÝ', 'AADEEILLNOOOTUUUY')"/>
</xsl:function>
```

Tuto úlohu nelze řešit pojmenovanou šablonou, neboť funkci `zw:key()` budeme potřebovat ve výrazu XPath.

Vlastní výstup bude proveden šablonou:

```
<xsl:template name='headers'>
  <xsl:param name='sorted-items' as='element()'
             select='$sorted-list'/>
  <xsl:text>\begin{theindex}</xsl:text>
  <xsl:for-each-group select='$sorted-items/*'
       group-adjacent="if (matches(@id, '^[^\p{L}]'))
       then $symbols else
       if ($use-ch and upper-case(substring(@id, 1, 2)) = 'CH')
```

```
      then 'CH'
      else zw:key(upper-case(substring(@id, 1, 1)))">
    <xsl:value-of select="concat('&#10;&#10;', $lethead-prefix,
                  current-grouping-key(), $lethead-suffix)"/>
    <xsl:apply-templates select='current-group()'/>
  </xsl:for-each-group>
  <xsl:text>&#10;\end{theindex}&#10;</xsl:text>
</xsl:template>
```

Formátování rejstříkových položek je dosaženo použitím týchž šablon jako v případě výstupu bez hlaviček.

2.4. Výhody modularizace

Zřejmou výhodou modularizace je možnost snadné změny formátování výstupu analogicky, jak to řeší MakeIndex ve stylových souborech. Můžeme například v kopii stylu `makeindex.xsl` předefinovat šablonu pro zápis položek.

Modularizace nabízí ještě další možnosti. Nemusíme totiž zpracovávat pouze rejstříky LaTeXových dokumentů. Styl `sort-index.xsl` ponecháme beze změny, ale vytvoříme si vlastní styl (či jiný program) pro konverzi do stejného formátu, jaký generuje styl `parse-index.xsl`, a styl pro formátování výstupu. Abyste si tyto programy mohli snadno otestovat, je struktura pomocných souborů popsána schématem Relax NG, jež bylo programem TRANG [9] konvertováno na XML Schema.

Vezměme si ještě jiný příklad. Předpokládejme, že máme knihu o botanice, k níž chceme sestavit rejstřík českých názvů, rejstřík latinských názvů, rejstřík anglických názvů, rejstříky názvů v dalších jazycích. Dále chceme rejstřík citovaných knih s čísly stran, kde se odkaz vyskytuje, rejstřík citovaných autorů – a pokud budeme takto pokračovat, můžeme překročit povolený počet otevřených souborů. Při použití implementace v XSLT vše vložíme do jediného souboru, pouze různé druhy položek označíme odlišnými makry. Abychom nemuseli tento soubor opakovaně načítat, přepíšeme si výstupní styl `format-index.xsl` tak, aby se každý typ setříděného rejstříku generoval vlastním elementem `<xsl:result-document>`, v němž zavoláme šablonu `sorted-index` se správnými parametry.

3. Náměty k reimplementaci BibTeXu

Myšlenka reimplementace BibTeXu pomocí nástrojů XML není nová (např. Widmann [10], Dagnat et al. [11], Hufflen [12], Beebe [13]). Popisované implementace jsou však založeny na specializovaných nástrojích, případně na jazyku, který

pouze připomíná XSLT. Reimplementace založená čistě na XSLT s využitím modulárních transformačních stylů by otevřela nové možnosti. Správa rozsáhlé bibliografické databáze v obyčejném souboru, ať už ve formátu `.bib`, nebo v XML, je dosti nepohodlná. Tyto údaje by mohly být uloženy v nějaké SQL databázi s pohodlným uživatelským rozhraním, k níž by se z XSLT přistupovalo pomocí XQuery [14], případně by údaje mohly být uloženy v databázi XIndice [15], jež pracuje přímo s XML. Taková implementace by jistě pomohla i projektu Biblet [16].

4. Závěr

České pořekadlo říká: „Kolik jazyků umíš, tolikrát jsi člověkem.“ V obecné rovině je pravdivé, ale z programátorského hlediska lze o jeho platnosti polemizovat. Zatímco s některými cizinci se domluvíme výhradně jejich mateřštinou, téhož výsledku lze dosáhnout programem napsaným v různých jazycích.

Ukázali jsme si, že program MakeIndex lze reimplementovat snadno použitím XSLT. Nepotřebujeme tedy specializovaný nástroj. Místo toho, abychom se učili řadu různých jazyků a jednoúčelových nástrojů, s využitím spolupráce TEXu či LATEXu s procesorem XSLT bychom více času mohli věnovat sazbě věcně správných a typograficky kvalitních dokumentů.

Reference

[1] P. Olšák: *Nový encTEX – kódování UTF-8 v TEXu*. Zpravodaj Československého sdružení uživatelů TEXu, **13** (2), 98–106 (2003).
[2] XSLT 2.0 – `http://www.w3.org/TR/xslt20/`
[3] XPath 2.0 – `http://www.w3.org/TR/xpath20/`
[4] M. Kay: *Schema-aware XSLT Processing*. Konference XML Prague, červen 2005. `http://www.xmlprague.cz`
[5] `http://saxon.sf.net/`
[6] `http://www.goldencode.com`
[7] `http://www.innotek.de`
[8] `http://icebearsoft.euweb.cz/xslt-indexing/`
[9] Trang: Multi-format schema converter based on RELAX NG – `http://www.thaiopensource.com/relaxng/trang.html`
[10] T. Widmann: *Bibulus—a Perl/XML replacement for BibTEX*. TUGboat 24 (3), 468–471 (2003).
[11] F. Dagnat, R. Keryell, L. B. Sastre, E. Donin de Rosière, N. Torneri: *BibTEX++: Toward higher-order BibTEXing*. TUGboat 24 (3), 472–488 (2003).

[12] J.-M. Hufflen: *European bibliography styles and MlBibTEX*. TUGboat 24 (3), 489–498 (2003).
[13] N. Beebe: *A bibliographer's toolbox*. TUGboat 25 (1), 89–104 (2004).
[14] XQuery 1.0 – `http://www.w3.org/TR/xquery/`
[15] Apache XIndice – `http://xml.apache.org/xindice/`
[16] T. Miller: *Biblet: A portable BibTEX bibliography style for generating highly customizable XHTML*. TUGboat 26 (1), 85–96 (2005).

Summary: Processing auxiliary TEX files with XSLT 2.0

The article shows possibilities of processing auxiliary TEX files with XSTL 2.0. The idea is demonstrated on reimplementation of MakeIndex in XSLT. Some thoughts concerning the possibilities of reimplementation of BibTEX purely in XSLT are also presented.

Schválené grantové projekty

Výbor CSTUGu schválil tři grantové projekty, které podle jeho mínění budou užitečné pro českou a slovenskou komunitu uživatelů TEXu. Návrhy grantů jsou zveřejněny v následujícím textu a podrobné informace lze nalézt na webových stránkách sdružení `http://www.cstug.cz`.

Návrh grantu — česká a slovenská podpora pro balík babel LaTeXu

Cíl:
V základní distribuci LaTeXu mít kvalitní podporu češtiny a slovenštiny v balíku babel a pro sazbu fonty v kódování T1, s vhodnou (ne nezbytně stoprocentní) mírou zpětné kompatibility se současnými způsoby sazby českých dokumentů (např. czech.sty z cstexu, csquote ap.).

Požadavky:
1) Upravit stávající verzi podpory české a slovenské sazby v distribuci babelu Johannese Braamse ve dvou fázích tak, aby se minimalizovaly problémy s užitím stylu.