Revisiting Lexicographical Order Relations on Person Names*

Jean-Michel HUFFLEN
LIFC (EA CNRS 4157)
University of Franche-Comté
16, route de Gray
25030 BESANÇON CEDEX
FRANCE
hufflen@lifc.univ-fcomte.fr
http://lifc.univ-fcomte.fr/~hufflen

Abstract

When a bibliography is built by extracting references from a data base — as BibTeX does when it builds a bibliography for a LaTeX document — these references are usually sorted w.r.t. author names. Nevertheless, most of implementations of lexical order relations on person names consist of concatenations of the parts of names: first name, particle, last name. We show that this may lead to incorrect results in some particular cases (some examples are given using BibTeX). Then we explain how this problem is solved in MlBibTeX, our multilingual reimplementation of BibTeX.

Keywords Lexicographical order relations, dictionaries, bibliographies, Unicode, XSLT 1.0, XSLT 2.0, MlBibTeX, nbst, Scheme.

Streszczenie

Gdy bibliografia jest budowana przez pobieranie odwołań z bazy danych — jak to robi BibTeX kiedy buduje bibliografię dla dokumentu LaTeX-owego — to odwołania są zwykle sortowane wg. nazwisk autorów. Zwykle implementacje porządku leksykograficznego według nazwisk osób są złożeniami składowych: imienia, partykuły, nazwiska. Pokazujemy, że może to w szczególnych przypadkach prowadzić do niepoprawnych wyników (podamy przykłady z użyciem BibTeX-a). Następnie wyjaśnimy, jak ten problem został rozwiązany w MlBibTeX-u, naszej językowo niezależnej reimplementacji BibTeX-a.

Słowa kluczowe Zasady sortowania leksykograficznego, słowniki, bibliografie, Unikod, XSLT 1.0, XSLT 2.0, MlBibTeX, nbst, Scheme.

0 Introduction

Given the bibliographical citations of a document's body, referring to *entries* of bibliography databases, a *bibliography processor*'s task consists of extracting the information concerning these entries, and arranging it in order for a word processor to be able to add all the bibliographical references as a 'References' section, usually put at a document's end. *Bibliography styles* control the layout of bibliographical references: for example, authors' first names are sometimes put *in extenso*, sometimes abbreviated. A good example of such a cooperation between a word and bibliography processor is given by LaTeX and BibTeX [12, § 12.1.3], this bibliography processor providing many bibliography styles and various types of bibliographical entries: `article`, `book`, `booklet`, etc. [12, Tables 13. 1 & 13.4].

Most often, references are sorted according to authors' or editors' names, even if there exist *unsorted* styles, that is, the order of items is the order of first citations of these items throughout the document. Concerning the sort operation w.r.t. 'authors' or editor's names', a document usable within a bibliography is supposed to be attributed to an *author*, except for some particular cases: an anthology or a conference's proceedings, in which case an *editor* is given. Some documents may be anonymous — good examples are given by Web pages — in which case a *key* is used for sorting. BibTeX uses this *modus operandi* — by means of a `KEY` field — for the entry

* Title in Polish: *Jeszcze raz o porządku leksykograficznym wg. osób*

```
robeson␣␣kenneth␣␣␣␣1964␣␣␣␣man of bronze
robeson␣␣kenneth␣␣␣␣2␣␣␣␣man of bronze
robeson␣␣kenneth␣␣␣murray␣␣will␣␣␣␣1992␣␣␣␣white eyes   ⟸ Kenneth Robeson and Will Murray
robeson␣␣kenneth␣␣␣et al␣␣␣␣1975␣␣␣␣king maker          ⟸ Kenneth Robeson and others
du bois␣␣paul␣␣␣␣2008␣␣␣␣title                           ⟸ first => Paul, von => du, last => Bois
du bois␣␣paul␣␣␣␣2008␣␣␣␣title                           ⟸ first => Paul, last => {Du Bois}
```

Figure 1: Examples of sort keys computed within BibTeX's standard bibliography styles.

types `booklet`, `manual`, `misc`. Moreover, this information often refers to a real person, but is sometimes given by an institution's name, viewed as a last name without a first name. In the following, the 'author' word will denote this kind of information, by language abuse.

The purpose of this article is to show that this sort operation may lead to strange results, so it should be specified precisely. In [7], we explained that sorting words is a language-dependent operation, and how we tackle this problem in MlBibTeX[1], our reimplementation of BibTeX focusing on multilingual features. The problem addressed here is to assemble partial results of this operation. In Section 1, we show how most of BibTeX's standard bibliography styles proceed. Then we mention that MlBibTeX's some new features cause this problem to be more complicated.

Let us recall that the result of parsing a bibliography database (.bib) file by MlBibTeX may be viewed as an XML[2] tree, according to the conventions of SXML[3]. That allowed us to use a language close to XSLT[4] — nbst[5] — for specifying bibliography styles. So we examine which solutions are provided by XSLT in Section 3. Finally, Section 4 explains the compromise we have reached in MlBibTeX.

Reading this article requires only a basic knowledge of BibTeX and XML. Some parts related to XSLT are more technical — especially some features related to XSLT 2.0, the new version — but should be understood after reading [8].

1 BibTeX's standard bibliography styles

A comprehensive study of the management of names in BibTeX — that is, values associated with the fields `AUTHOR` and `EDITOR` — is given in [6]. Here we just recall that BibTeX recognizes four components inside a name: *First* (for a first name), *von* (for a particle), *Last* (for a last name), *Junior* [13, § 4]. As suggested by the capitalisation used within this terminology, the words belonging to the *von* field are supposed to begin with a lowercase character, whereas the words belonging to the *First* and *Last* fields are supposed to begin with an uppercase character.

The `SORT` command used in BibTeX's bibliography styles [12, Table 13.7] is based on sort keys computed for each entry, and stored into *entry variables* — existing for each entry — `sort.key$`. If we look into standard bibliography styles, we can see that this sort key for an entry is mainly based on concatenating parts of authors' names, followed by the entry's year and the title[6]. This string is truncated over `entry.max$` characters[7].

Some examples of sort keys are given in Figure 1. Sort keys are based on concatenations of string using only digits and lowercase characters. Non-alphanumeric characters are removed by means of BibTeX's `purify$` function [12, Table 13.8]. We can notice that comparison levels are denoted by different numbers of consecutive space characters: one space character may appear inside a part of a name[8], two consecutive space characters separate the *Last* and *First* parts of the same name, three (resp. four) consecutive space characters appear before a new name (resp. the year and the title).

A first remark: within most of BibTeX's standard bibliography styles, the *von* and *Last* part are separated only by a single space character within `sort.key$`'s values. That causes a name to be alphabeticised w.r.t. the particle, if it exists. In particular, Figure 1 shows that the names '`du Bois, Paul`' and '`Du Bois, Paul`' are viewed equivalent. That is probably American usage, but according to

[1] **M**ulti-**L**ingual BibTeX.

[2] e**X**tensible **M**arkup **L**anguage. Readers interested in an introductory book to this formalism can refer to [14].

[3] Scheme implementation of XML. See [10] for more details.

[4] e**X**tensible **S**tylesheet **L**anguage **T**ransformations, the language of transformations used for XML texts.

[5] **N**ew **B**ibliography **ST**yles.

[6] Some initial words irrelevant for a 'semantic' sort operation are also removed: '`A`', '`An`', '`The`'.

[7] BibTeX's `entry.max$` variable is bound to 250.

[8] When BibTeX parses the value of a field, several consecutive occurrences of this character are replaced by a single one.

European background, '`du Bois, Paul`' should be alphabeticised under '`B-`', as 'Bois (Paul du)'. However, this behaviour could be fixed easily in this case by changing the order of parts within the concatenations performed for a name.

The '`others`' keyword is replaced by '`et al`' inside a sort key. As a consequence, the `AUTHOR` information '`Kenneth Robeson and others`' comes before '`Kenneth Robeson and Will Murray`'. We think that is incorrect: an ellipsis about additional names should be ranked after specified additional names. In this case, too, that could be fixed by another replacement value for '`others`'. Another point is that years are compared lexicographically, so `2` comes after `1964`! Of course, this point is not very important in practice because years coming from 'actual' bibliography database files are often close each to others; it is rare to include entries for documents written in the 1st and 20th centuries, but in such a case, the sort operation fails[9].

Besides, let us notice that sorting w.r.t. month information must be explicitly programmed in bibliography styles, it does not appear within sort keys for most standard styles. In fact, nothing is specified about entries sharing the same sort key: according to such styles, the only way to influence the order of items sharing the same author and year information is to add dummy commands at the beginning of the value associated with a `TITLE` field:

```
TITLE = {\before The Man of Bronze}
TITLE = {\last Brand of the Werewolf}
```

Since these dummy commands — equivalent to the `\relax` command [11, Ch. 24] — must be defined when LaTeX processes the generated references, such a workaround complicates the sharing of such entries among several people.

To sum up, the sorting operation performed by BibTeX works in most practical cases, but not always. Some points are quite easy to customise, some are more difficult to fix, and some are unfortunately hard-wired.

2 Improving BibTeX

MlBibTeX's syntactical improvements about specifying names are described in [6]. Here we just recall that co-authors are introduced by the '`and`' keyword, like in BibTeX, and possibly followed by *collaborators*[10], introduced by the '`with`' keyword. A good example is given by the co-authors and collaborators of *The LaTeX Companion*'s second edition [12]:

```
AUTHOR = {Frank Mittelbach and
          Michel Goossens with
          Johannes Braams with
          David Carlisle with
          Chris A. Rowley with
          Christine Detig with
          Joachim Schrod}
```

Like in BibTeX, the '`others`' keyword can be used: '`and others`' (resp. '`with others`') for additional co-authors (resp. collaborators) left unspecified.

The specification of `author` and `editor` elements within the representation of bibliographical entries in XML used internally by MlBibTeX is given in Figure 2. Initially, this DTD[11] was derived from [2, § B.4.4.3] and has been extended to all the elements and attributes used throughout MlBibTeX. However, for sake of simplicity, we have dropped out some possible children of the `name` element[12].

In [4], we give a simple example of sorting bibliographical items, provided that there is only an author. In fact, the actual template uses an external function written in Scheme[13]. The problem is more complicated because the maximum number of possible authors is not bound *a priori*, and because there are two connectors: `and`, `with`. Moreover, we cannot mix co-authors and collaborators; the latter should be used as additional sort keys, for bibliographical items sharing the same sequences of co-authors.

3 Using XSLT

Now let us examine how sorting person names can be put into action using XSLT. If we consider XSLT 1.0 [17, § 10], an acceptable solution is probably the concatenation of all the parts of a name, as did in BibTeX, since the only way to get a sort key is the

[9] This error disappears if '`2`' is replaced by '`0002`' in the value associated with the `YEAR` field. But that causes '`0002`' to be put down in the generated reference processed by LaTeX, so that is only a workaround.

[10] This notion of collaborators also exists in the bibliographies built with DocBook, an XML system for writing structured documents [19].

[11] **D**ocument **T**ype **D**efinition. Such a file defines a document markup model, see [14, pp. 148–155] for more details. Now *schemas* are more and more used for such a definition, but we would not take any actual advantage of them for our present purpose.

[12] Such elements are used for multilingual purposes. For example, when an author is expressed using another language than the current entry's, e.g.:

```
AUTHOR = {[Robert Silverberg] : english}
```

whereas the language's entry is `french`. Another use concerns possible transliteration of names originating from languages using non-Latin alphabets:

```
AUTHOR =
  {[Александр Константинович Глазунов] * russian
   [Alexander Konstantinovich Glazunov]}
```

See [6] for more details.

[13] MlBibTeX is written using Scheme. Readers interested in an introductory book to this functional programming language can refer to [15].

```
<!-- Using entity parameters for repeated specifications. -->
<!ENTITY % author-or-editor   "(name,(and,name)*,and-others?,(with,name)*,with-others?)">
<!ENTITY % language-possibly  "language NMTOKEN #IMPLIED">
<!-- Authors and editors. -->
<!ELEMENT author   %author-or-editor;>
<!ELEMENT editor   %author-or-editor;>
<!ELEMENT name     (personname | othername | ...)>   <!-- Other elements are used for multilingual
                                                         purposes.
                                                    -->
<!-- Person names and organisation names used as authors or editors. -->
<!ELEMENT personname   "(first?,von?,last,junior?)">   <!-- Parts originating from BibTeX. -->
<!ATTLIST personname   %language-possibly;>            <!-- When the name's language is not the
                                                         entry's.
                                                      -->
<!ELEMENT othername    "(#PCDATA | asitis | emph)*">
                       <!-- An asitis element remains insensitive to any case change ordered by
                            bibliography styles. An emph element expresses stylistic information, e.g., using
                            italicised characters.
                         -->
<!ATTLIST othername    %language-possibly;
                       "sortingkey CDATA #IMPLIED"/>
                       <!-- Since othername elements may contain markup or words irrelevant inside a sort
                            key, this key can be redefined as an attribute's value.
                         -->
<!-- Names' parts. -->
<!ELEMENT first         (#PCDATA)>
<!ATTLIST first         abbrev CDATA #IMPLIED>   <!-- When a first name is not abbreviated using a -->
<!ELEMENT von           (#PCDATA)>               <!-- 'standard' way. -->
<!ELEMENT last          (#PCDATA)>
<!ELEMENT junior        (#PCDATA)>
<!ELEMENT and           EMPTY>                   <!-- Between co-authors. -->
<!ELEMENT with          EMPTY>                   <!-- Before a collaborator's name. -->
<!ELEMENT and-others    EMPTY>                   <!-- And more co-authors. -->
<!ELEMENT with-others   EMPTY>                   <!-- And more collaborators. -->
```

Figure 2: Our conventions for authors and editors.

use of the `select` attribute of the `xsl:sort` element. If there are several authors, there cannot be as many `xsl:sort` elements — giving primary sort key, secondary sort key, etc. — as authors, since authors' number is not known statically. For the same reason, a complete concatenation of parts of all the names can only be implemented by means of an *extension function*, using another programming language than XSLT.

The situation is better in XSLT 2.0 [18, § 13], since a sort key given by means of the `select` attribute can be computed using an XPath 2.0's expression [8]. Let us consider person names expressed using the DTD given in Figure 2, then Figure 3 shows what to do if there is only one author or editor.

Unfortunately, this *modus operandi* cannot be generalised to multiple authors. Successive sort keys must be expressed using successive `xsl:sort` elements, so there is no way to insert some tests in order to check whether elements `and`, `and-others`, `with`, `with-others` elements are remaining. In fact, we experienced a solution, but it consists of a complete re-programming of the sort operation, using sequence constructors of XSLT 2.0 [18].

A solution working with the `xsl:sort` element is to build a concatenation of all the names, the elements `and`, `and-others`, `with`, `with-others` being replaced by markers belonging to the private use area of Unicode's basic multilingual plane [16]. To do that, we use characters entities like in [8, § 6].

```
...
<xsl:stylesheet version="2.0" ... xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:xsd="http://www.w3.org/2001/XMLSchema" ...>

  <xsl:template match="mlbiblio">   <!--  Root element of a bibliography.  -->
    <xsl:apply-templates>
      <xsl:sort
        select=
        "for $the-people in if (author) then author else editor return
             if ($the-people/name[1]/personname) then $the-people/name[1]/personname/last else
                if ($the-people/name[1]/othername) then
                   $the-people/name[1]/othername/(if (@sortingkey) then @sortingkey else .)  else
                   '' (:  Some other cases are dropped out (cf. Fig. 2).  :)"/>
      <xsl:sort select="(if (author) then author else editor)/name[1]/personname/first"/>
      <xsl:sort select="(if (author) then author else editor)/name[1]/personname/von"/>
      <xsl:sort select="(if (author) then author else editor)/name[1]/personname/junior"/>
      <xsl:sort select="xsd:integer(year)"/>     <!--  Secondary sort key (numerical sort).  -->
      <xsl:sort select="add:month-position(month)"/>
      <!--  The add:month-position function is given in [8, Fig. 5].  -->
    </xsl:apply-templates>
  </xsl:template>

  ...
</xsl:stylesheet>
```

Figure 3: Using several sort keys when there is only one author or editor.

The result is given in Figure 4. We intentionally put as many '`as`' attributes as possible, these attributes specifying type information (cf. [8, § 5]), in order for readers to see more easily which type is used by each variable, which type is returned by each template computing a part of the primary sort key. We use a *mode* [18, § 6.5] for computing sort keys, templates without modes are reserved for putting down the contents of generated references, that is, the result of this stylesheet.

Michael Kay [9, p. 429] mentions that such an implementation, based on concatenations, is preferable. As an example, it sorts the name '`Macarthur, John`' before '`MacArthur, Philip`'. Using different sort keys for the last and first names would revert this order, because a tertiary difference — the case — in the last name is considered more significant that a primary difference — the characters — in the first name[14]. From our viewpoint, this point is debatable and should anyway be decided by bibliography style designers. In addition, if we consider efficiency, building concatenations causes much space to be allocated, and many character sequences to be copied, whereas examining the first letters of the last name occurring at first often makes a difference. Of course, the same remark holds good about the string concatenations performed by BibTeX's standard bibliography styles (cf. § 1).

[14] See [7] for a more complete explanation about these successive steps of a lexicographic order among strings.

4 MlBibTeX's solutions

The revisions we propose hereafter should be viewed as compromises: MlBibTeX can be used as it is, and work 'as well as BibTeX'. That is, users can accept the results of the default sorting operation[15]. But these revisions should maintain a 'classical' use of the `nbst:sort` element, and allow users to perform a better customisation about sorting bibliographical items w.r.t. authors' names.

A new field, so-called `LASTSORTKEY`, has been added to the fields recognised by MlBibTeX. This field is optional, and must be set to an integer (possibly negative). It is modelled as an attribute in our XML representation:

```
<article lastsortkey="...">...</article>
<book lastsortkey="...">...</book>
...
```

and can be used in the last step of a sort operation, as did in 'new' standard bibliography styles

[15] Concerning us, we were processing some tests between the compatibility mode for 'old' bibliography styles [5] and were puzzled because `bst`'s sort and `nbst`'s did not result in the same order. That was the story's beginning...

```
...
<!DOCTYPE stylesheet [<!ENTITY start-firstname "&#xE000;">
                      <!ENTITY start-von "&#xE001;">
                      <!ENTITY start-junior "&#xE002;">
                      <!ENTITY and-marker "&#xE003;">
                      <!ENTITY and-others-marker "&#xE004;">
                      <!ENTITY with-marker "&#xE005;">
                      <!ENTITY with-others-marker "&#xE006;">]>
<xsl:stylesheet version="2.0" ... xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:xsd="http://www.w3.org/2001/XMLSchema" ...>
  <!-- Putting bibliographical items down, w.r.t. the 'classical' order. -->
  <xsl:template match="mlbiblio">
    <xsl:apply-templates>
      <xsl:sort>     <!-- Primary sort key. -->
        <xsl:variable name="sort-people-subkey-s" as="xsd:string+">
          <xsl:apply-templates select="(if (author) then author else editor)/*"
                               mode="sort-people-key"/>
        </xsl:variable>
        <xsl:value-of select="$sort-people-subkey-s" separator=""/>
      </xsl:sort>
      <xsl:sort select="xsd:integer(year)"/>
      <xsl:sort select="add:month-position(month)"/>
    </xsl:apply-templates>
  </xsl:template>
  <!-- Computing the sort key for a name, resulting in a single string. -->
  <xsl:template match="name" mode="sort-people-key" as="xsd:string">
    <xsl:apply-templates mode="sort-people-key"/>
  </xsl:template>
  <xsl:template match="personname" mode="sort-people-key" as="xsd:string">
    <xsl:value-of select="last,'&start-firstname;',first,'&start-von;',von,'&start-junior;',junior"
                  separator=""/>
  </xsl:template>
  <xsl:template match="othername" mode="sort-people-key" as="xsd:string">
    <xsl:value-of select="."/>
  </xsl:template>
  <!-- Replacing connector elements by markers. In the first two cases, xsl:text elements are useless, because
       markup surrounds the string to be put down.
    -->
  <xsl:template match="and" mode="sort-people-key" as="xsd:string">&and-marker;</xsl:template>
  <xsl:template match="with" mode="sort-people-key" as="xsd:string">&with-marker;</xsl:template>
  <xsl:template match="and-others" mode="sort-people-key" as="xsd:string">
    <xsl:text>&and-others-marker;</xsl:text>
  </xsl:template>
  <xsl:template match="with-others" mode="sort-people-key" as="xsd:string">
    <xsl:text>&with-others-marker;</xsl:text>
  </xsl:template>
  ...
</xsl:stylesheet>
```

Figure 4: Sorting keys using concatenation.

(cf. Fig. 6). It is especially useful to sort such items sharing the same authors' names, the same year, and the same month. A missing `LASTSORTKEY` value takes precedence over a present one. If several entries share the same sort key, including the same value associated with the `LASTSORTKEY` fields[16], the original order is retained[17]. That means that if the `\bibliography` command of a LaTeX source text is:

`\bibliography{...,`f_i`,...,`f_j`,...}`

and let e_x and e_y be two bibliographical items sharing the same sort key. If e_x (resp. e_y) comes from the .bib file f_i (resp. f_j), then e_x comes before e_y within the generated bibliography. The same if e_x takes precedence over e_y within the same .bib file. Let us recall that superfluous fields are ignored by 'old' BibTeX, so this field can be added without disturbing this program. In addition, since BibTeX's standard bibliography styles ignore this new field, it is also ignored when these styles are applied by means of MlBibTeX's compatibility mode [5].

The second change concerns the `nbst:sort` element. Its original definition [3, App. A] makes it very close to XSLT 1.0's [17, § 10], but not identical. Like this `xsl:sort` element coming from Version 1.0, it provides insufficient service, unless if it used with functions written using a 'more classical' programming language. We think that some simple functionalities — like finding a month name's position — can be programmed using Scheme, but specifying a crucial operation such as sorting bibliographical items w.r.t. names should not depend on deep knowledge of Scheme. In other words, we think that we cannot require that a style designer should be a Scheme expert. Some operations, such as language-dependent lexicographical order relations are to be programmed in Scheme, but we tried to reach a form easily understandable by basic programmers [7].

What about a new element, close to XSLT 2.0 [18, § 13]? More generally, why nbst would not be close to XSLT 2.0? That would cause major rewriting even if such an evolution could be a good idea. However, it would be a partial solution since we think that a sort operation based on the concatenation of parts of all the names is not really efficient (cf. § 3).

The compromise is an extended definition of the `nbst:sort` element, given in Figure 5. This new definition overrides the old one, given in [3, App. A]. The new element works as follows.

[16] ... or if the values associated with this field are both missing.

[17] In programming's terminology, such a sort is called *stable* sort.

```
<nbst:sort
    select=expr language=lg-idf
    data-type=("text" | "number")
    order=("ascending" | "descending")
    case-order=("upper-first" | "lower-first")
    use=name personname-part-order=part-order
    and-as=code-list and-others-as=code-list
    with-as=code-list with-others-as=code-list>
  template
</nbst:sort>
```

where:

code-list a space-separated list whose elements are natural numbers,

expr is analogous to an XPath expression,

lg-idf a language identifier,

name an identifier,

part-order a space-separated list whose elements are `first`, `junior`, `last`, `von`,

template a (possibly empty) sequence of nbst elements, except for the top-level ones.

Default values are underlined.

Figure 5: New `nbst:sort` element in nbst.

- Invoking *template* yields the sort key, except if the `select` attribute gives it, in which case *template* must be empty. If both *template* and *select* are absent, this is equivalent to specifying '`select="."`', that is, the sort key is the identity function.
- If the `use` attribute is given, all the other attributes, except for `select`, are irrelevant and cause errors. The value associated with the `use` attribute must be a Scheme function whose model is:

  ```
  (define (a-function-name rel?)
    (lambda (node-0 node-1 k0)
      ...))
  ```

  where:
  - *rel?* receives the order relation associated with the bibliography's language, e.g., a Scheme function given in [7, Fig. 2];
  - *node-0* and *node-1* are two SXML nodes;
  - *k0* receives the function implementing the next sort key, that is, the function to be called when two nodes are equal w.r.t. the present relation[18].

  The function resulting from evaluating the expression `(a-function-name rel?)` should return `#t` (a 'true' value) if *node-0* is to be put

[18] In functional programming, such an argument is used within the *Continuation-Passing Style* [1].

before *node-1*, `#f` (the 'false' value) if *node-0* is to be put after *node-1*; otherwise the *k0* function is applied and allows us to process the next sort key, that is, the next `nbst:sort` element.

- If the `use` attribute is absent, we look for the attributes:
  - `personname-part-order`, giving the primary sort key, secondary sort key, etc. for a `personname` element,
  - `and-as`, `and-others-as`, `with-as`, and `with-others-as`, whose associated values are viewed as successive codes of the characters of a 'dummy' string. Let us recall that the MlBibTeX's current version is based on Latin 1 encoding [7], so the codes of 'actual' characters are less than 256. If you want these markers to be viewed as characters greater than 'actual' characters, put natural numbers greater than or equal to 256.

  Of course, these attributes will work if the sort key is either an `author` or an `editor` element. In another case, generated functions will always return a 'true' value and the node set will remain in the original order. In addition, let us notice that the `data-type` attribute must be set to `text` — its default value — in this case. If a subpart of these attributes is only provided, the omitted ones default to an empty list,
- If none of these attributes:

`use`	`and-others-as`
`personname-part-order`	`with-as`
`and-as`	`with-others-as`

  is given, the `data-type` attribute may be set to `number` (resp. `text`) for a lexicographical (resp. numerical) sort.
- The meaning of the other attributes — `order`, `case-order` — is unchanged.

Most of bibliography styles coming as part of MlBibTeX's source files use this nbst:sort element as shown in Figure 6.

5 Conclusion

As mentioned in [6], dealing with person names is a difficult problem, since we have to face many figure cases. As mentioned in [9, p. 429], sorting person names can be defined carefully. And let us not forget that this order is language-dependent [7]. We think that MlBibTeX provides a good and complete toolbox to tackle this problem and put acceptable solutions into action. But we would not be surprised if a new version had to refine these tools. We only hope that such refinement will be slight.

```
<nbst:template match="mlbiblio">
  ...
  <nbst:apply-templates>
    <nbst:sort use="&lt;authors&lt;?"/>
    <nbst:sort select="year"
               data-type="number"/>
    <nbst:sort
      select="call(month-position,month)"
      data-type="number"/>
    <nbst:sort>
      <nbst:choose>
        <nbst:when test="@lastsortkey">
          <nbst:value-of select="@lastsortkey">
        </nbst:when>
        <nbst:otherwise>
          <nbst:value-of select="-Inf"/>
        </nbst:otherwise>
      </nbst:choose>
    </nbst:sort>
  </nbst:apply-templates>
  ...
</nbst:template>
```

where:

- `<authors<?` is a Scheme function provided within the source files of MlBibTeX, it efficiently compares SXML representations of bibliographical items — `article`, `book`, `booklet`, ... and other children of the `mlbiblio` root element — w.r.t. `author` or `editors` subtrees by using as sort keys as needed;
- `month-position` is also a Scheme function provided by MlBibTeX that returns the same result than the `add:month-position` function written in XSLT in [8, Fig. 5] — there is no `nbst:function` element in `nbst` —;
- the '`-Inf`' expression returns the smallest negative integer.

Figure 6: 'Standard' use of the `nbst:sort` element.

6 Acknowledgements

Many thanks to Jerzy B. Ludwichowski, who has written the Polish translation of the abstract.

References

[1] Daniel P. Friedman, Mitchell Wand and Christopher T. Haynes: *Essentials of Programming Languages*. The mit Press. 1992.

[2] Michel Goossens and Sebastian Rahtz, with Eitan M. Gurari, Ross Moore and Robert S. Sutor: *The LaTeX Web Companion*. Addison-Wesley Longmann, Inc., Reading, Massachusetts. May 1999.

[3] Jean-Michel Hufflen: "MlBibTeX's Version 1.3". TUGboat, Vol. 24, no. 2, pp. 249–262. July 2003.

[4] Jean-Michel Hufflen: "Bibliography Styles Easier with MlBibTeX". In: *Proc. EuroTeX 2005*, pp. 179–192. Pont-à Mousson, France. March 2005.

[5] Jean-Michel Hufflen: "BibTeX, MlBibTeX and Bibliography Styles". *Biuletyn* GUST, Vol. 23, pp. 76–80. In *BachoTeX 2006 conference*. April 2006.

[6] Jean-Michel Hufflen: "Names in BibTeX and MlBibTeX". TUGboat, Vol. 27, no. 2, pp. 243–253. TUG 2006 proceedings, Marrakesh, Morocco. November 2006.

[7] Jean-Michel Hufflen: "Managing Order Relations in MlBibTeX". TUGboat, Vol. 29, no. 1, pp. 101–108. EuroBachoTeX 2007 proceedings. 2007.

[8] Jean-Michel Hufflen: "XSLT 2.0 vs XSLT 1.0". In: *this volume*. BachoTeX. April 2008.

[9] Michael H. Kay: XSLT *2.0 Programmer's Reference*. 3rd edition. Wiley Publishing, Inc. 2004.

[10] Oleg E. Kiselyov: XML *and Scheme*. September 2005. `http://okmij.org/ftp/Scheme/xml.html`.

[11] Donald Ervin Knuth: *Computers & Typesetting. Vol. A: The TeXbook*. Addison-Wesley Publishing Company, Reading, Massachusetts. 1984.

[12] Frank Mittelbach and Michel Goossens, with Johannes Braams, David Carlisle, Chris A. Rowley, Christine Detig and Joachim Schrod: *The LaTeX Companion*. 2nd edition. Addison-Wesley Publishing Company, Reading, Massachusetts. August 2004.

[13] Oren Patashnik: *BibTeXing*. February 1988. Part of the BibTeX distribution.

[14] Erik T. Ray: *Learning* XML. O'Reilly & Associates, Inc. January 2001.

[15] George Springer and Daniel P. Friedman: *Scheme and the Art of Programming*. The MIT Press, McGraw-Hill Book Company. 1989.

[16] The Unicode Consortium: *The Unicode Standard Version 5.0*. Addison-Wesley. November 2006.

[17] W3C: XSL *Transformations (*XSLT*). Version 1.0*. W3C Recommendation. Edited by James Clark. November 1999. `http://www.w3.org/TR/1999/REC-xslt-19991116`.

[18] W3C: XSL *Transformations (*XSLT*). Version 2.0*. W3C Recommendation. Edited by Michael H. Kay. January 2007. `http://www.w3.org/TR/2007/WD-xslt20-20070123`.

[19] Norman Walsh and Leonard Muellner: *DocBook: The Definitive Guide*. O'Reilly & Associates, Inc. October 1999.