Introduction to XQuery and XPath Full Text*

Jean-Michel HUFFLEN
LIFC (EA CNRS 6942)
University of Franche-Comté
16, route de Gray
25030 BESANÇON CEDEX
FRANCE
`jmhufflen@lifc.univ-fcomte.fr`
`http://lifc.univ-fcomte.fr/home/~jmhufflen`

Abstract

We show how XQuery and XPath Full Text extends XQuery's basis and eases some applications related to advanced search of a text. Most examples we give yield LaTeX source texts.
Keywords XML, XQuery (1.0 & 1.1), XPath 2.0, full text search, substring search, generating (LA)TeX source texts.

Streszczenie

Pokażemy, jak „XQuery and XPath Full Text“ rozszerzają XQuery ułatwiając realizację pewnych zadań związanych z zaawansowanym przeszukiwaniem tekstów. Większość pokazanych przykładów generuje LaTeXowe teksty źródłowe.
Słowa kluczowe XML, XQuery (1.0 & 1.1), XPath 2.0, wyszukiwanie pełnotekstowe, wyszukiwanie według podłańcucha, generowanie tekstów źródłowych dla TeXa i LaTeXa.

0 Introduction

This present article follows the general introduction to XQuery given at the BachoTeX 2009 conference [4]. Let us recall that this language is a query language for data stored using XML[1] form. XQuery extends the most recent version of XPath (2.0) [9] — the language used to address parts of an XML document — and can be used to search such documents and arrange the result, as an XML structure or a simple text (possibly suitable for a TeX engine). 'Search of such documents' often reads '*substring* search', so in a first section, we explain why *full-text* search is different from substring search. Then we describe the main features of XQuery and XPath Full Text in Section 2. Last, more examples show that full-text search may be useful when source texts for (LA)TeX are derived. Reading this article only requires basic knowledge about XQuery, that is, about the features introduced in [4]. Of course, this article does not aim to replace the reference document of the W3C[2] [15]. The processor we use for the demonstration at the BachoTeX 2010 conference is BaseX [1]. Another interesting implementation is GalaTeX [2], but it uses old syntax for full-text selection[3].

1 Full-text search vs substring search

As sketched in [16, pp. 285–286], a full-text search performs a search for *tokens*[4] and *phrases* rather than substrings. For example, a 'classical' substring search for items that contain the string `"lease"` will return, in particular, an item containing the string:

```
"\TeX\ Live 2009 has been released"
```

A full-text search for the token `"lease"` will not. On the contrary, a full-text search should be support language-based searches including tokens with

* Title in Polish: *Wprowadzenie do „XQuery i XPath* Full Text“.

[1] **eX**tensible **M**arkup **L**anguage. Readers interested in an introductory book to this formalism can refer to [8].

[2] **W**ord **W**ide **W**eb **C**onsortium.

[3] This syntax is described in [13]. It is still accepted by the BaseX processor but only for sake of compatibility with previous versions.

[4] Informally, **tokenisation** breaks a character string into a sequence of tokens, units of punctuation, and spaces. W.r.t. the terminology used within reference documents about XQuery and XPath Full Text, a **token** is a non-empty sequence of characters returned by a tokeniser as a basic unit to be searched; a **phrase** is an ordered sequence of any number of tokens. [15] makes precise that these notions should be implementation-defined. See [15, § 4.1] for more details.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<books>
  <omnibus series="Doc Savage">
    <author>
      <name><personname><first>Kenneth</first><last>Robeson</last></personname></name>
    </author>
    <booktitle>Doc Savage Omnibus #9</booktitle>
    <year>1989</year>
    <story><title>The Invisible-Box Murders</title><year>1941</year></story>
    <story><title>Birds of Death</title><year>1941</year></story>
    <story><title>The Wee Ones</title><year>1945</year></story>
    <story><title>Terror Takes 7</title><year>1945</year></story>
  </omnibus>
  <omnibus series="Doc Savage">
    <author>
      <name><personname><first>Kenneth</first><last>Robeson</last></personname></name>
    </author>
    <booktitle>Doc Savage Omnibus #10</booktitle>
    <year>1989</year>
    <story><title>The Devil's Black Rock</title><year>1942</year></story>
    <story><title>Waves of Death</title><year>1943</year></story>
    <story><title>The Too-Wise Owl</title><year>1942</year></story>
    <story><title>Terror and the Lonely Widow</title><year>1945</year></story>
  </omnibus>
</books>
```

Figure 1: Specification of some stories collected in omnibus volumes.

the same linguistic stem. For example, if we consider the Polish declensions[5], searching the tokens with the same linguistic stem than '*robot*' (nominative singular) should find the other possible cases of this word[6], such as '*roboty*' (nominative plural) and '*robotów*' (genitive plural, as in '*Bajki robotów*'). Another example is given by the English language: finding the tokens with the same linguistic stem as 'mouse' should resulting in looking for 'mouse' and 'mice'. Last, full-text search may be inexact, so there is a notion of *score* or *relevance*. However, the reference document does not make precise this notion [15, § 2.3], so values used as scores — decimal numbers belonging to the range $[0, 1]$ — are implementation-dependent.

2 XQuery and XPath Full Text's features

Let us consider the XML text given in Fig. 1, already used in [4]. A 'classical' XPath 2.0 expression returning the `story` elements whose title contains the word 'Terror' is:

```
for $story-0 in
  doc(...)/books/omnibus/
  story[contains(data(title),"Terror")]
return $story-0
```

these `story` elements being:

```
<story>
  <title>Terror Takes 7</title>
  <year>1945</year>
</story>
<story>
  <title>Terror and the Lonely Widow</title>
  <year>1945</year>
</story>
```

The same result can be got by a full-text selection extending XQuery capabilities as follows[7]:

```
for $story-0 as element(story) in
  doc(...)/books/omnibus/story
where $story-0[title contains text "Terror"]
return ("\processstory{",$story-0,"}")
```

where `\processstory` is a (LA)TEX command able to typeset a `story` element. If you would like a case-insensitive search, just make precise:

[5] ... but unfortunately, no XQuery and XPath Full Text processor provides support for the Polish language now, as far as we know.

[6] When a language uses *declensions*, a word's function is directly expressed within this word, most often by a suffix. As examples, the *nominative case* marks the subject of a verb, the *genitive case* marks a noun as modifying another noun, most often as being its possessor.

[7] According to the old syntax described in [13], the constraint is written '`[title ftcontains "Terror"]`'.

```
for $story-0 in doc(...)/books/omnibus/story
where $story-0[title contains text "terror"
               using case insensitive]
return ("\processstory{",$story-0,"}")
```

'`using case sensitive`' being the default. Several full-text selections can be connected by means of the keywords `ftor` or `ftand`, whose semantics are respectively 'logical and' and 'logical or':

```
for $story-0 in doc(...)/books/omnibus/story
where $story-0[title contains text
               "Terror" ftor "7"]
return ("\processstory{",$story-0,"}")
```

There also exists an unary operator `ftnot`, in which case, matched texts do not satisfy the operand full-text selection. *Sets* can also be used:

```
for $story-0 in doc(...)/books/omnibus/story
where $story-0[title contains text
               { "Terror", "7" }]
return ("\processstory{",$story-0,"}")
```

which is equivalent to:

```
for $story-0 in doc(...)/books/omnibus/story
where $story-0[title contains text
               { "Terror", "7" } any]
return ("\processstory{",$story-0,"}")
```

that is, at least one word must appeared in the title's contents of a selected `story` element. If you would like all the words of a set to appear within such a title's contents, you can use:

```
for $story-0 in doc(...)/books/omnibus/story
where $story-0[title contains text
               { "Terror", "7" } all]
return ("\processstory{",$story-0,"}")
```

If you would like the words of a set to be in succession, use:

```
for $story-0 in doc(...)/books/omnibus/story
where $story-0[title contains text
               { "Terror", "and" } phrase]
return ("\processstory{",$story-0,"}")
```

which will yield the right answer whatever whitespace characters there are between these two words. As mentioned in § 1, you can look for tokens that have the same stem as the tokens and phrases written within the query — these tokens and phrases are supposed to be in Polish — :

```
for $s0 in ...
where $s0[. contains text "robot"
          using language "po" using stemming]
return $s0
```

As mentioned in § 1, too, the relevance of a selection can be measured by means of scores:

```
for $story-0 in doc(...)/books/omnibus/story
let score $s :=
  $story-0[title contains text { "Birds", "of" }
return $s
```

As another interesting feature, diacritics can be ignored:

```
for $s0 in ...
where $s0[. contains text contains "Krakow"
          using diacritics insensitive]
return $s0
```

that is, we look for 'Krakow' as well as 'Kraków'. *Stop words* can be substituted by any token, for example:

```
for $story-0 in doc(...)/books/omnibus/story
where $story-0[title contains text "in Death"
               using stop words("in")]
return ("\processstory{",$story-0,"}")
```

selects the `story` elements whose titles are *Birds of Death* and *Waves of Death*. In both cases, the stop word 'in' is substituted by the word 'of'.

XQuery and XPath Full Text 1.0 includes many other features we do not describe in detail here: using regular expressions (*wildcards*), constraints on the number of occurrences about a full-text selection, constraints on the distance — expressed by a number of words — between two items of a full-text selection. Note that these *options* — w.r.t. the terminology used within [15] — may or may not be implemented by XQuery and XPath Full Text processors. Last, let us mention that XML-like syntax exists: XQueryX for XQuery and XPath Full Text [15, App. E].

3 Deriving (LA)TEX source texts

We show how to build source texts for (LA)TEX from XML files by means of XSLT[8] in [3] and by means of XQuery in [4]. A short comparison between these two methods is sketched in [4] and a more general synthesis is given in [5]. If you are interested in processing textual contents of XML elements or values associated with attributes, XPath 2.0 and XSLT 2.0 only provide basic tools: the `tokenize` function of XPath [6, pp. 434–436], and the `analyze-string` element of XSLT [7, pp. 178–184]. So the features provided by XQuery and XPath Full Text are more suitable. However, there are two drawbacks: first, the character maps of XSLT 2.0, that allow output text to be written according to TEX's syntax for special characters, are implementation-dependent in XQuery and many XQuery processors do not propose them; second, XQuery and XPath Full Text's present version is built out of XQuery 1.0 [10], so the new features of XQuery 1.1 [14] — especially the groups of selected items — are unusable.

[8] e**X**tensible **L**anguage **S**tylesheet **T**ransformations [11].

We personally experienced XQuery and XPath Full Text with a medium-sized example. At the University of Franche-Comté, we manage the projets done by students in Computer Science. The master file grouping the whole information related to these projects — descriptions, supervisors, students' grades, ... — is written using XML-like syntax. Give some topics, we are able to find projects matching them, we can also know if these projects have succeeded.

4 Conclusion

The W3C proposes two extensions to XQuery. The first, XQuery Update Facility [12], allows an XML text to be changed and is obviously out of scope of any interface with LaTeX. The second, XQuery and XPath Full Text, allows information structured by XML elements and attributes to be searched by means of 'basic' XQuery's features. This second extension to basic features also allows textual information to be searched efficiently by means of Full-Text's features. Then selected information can be sent to a word processor. That is, XQuery, Full-Text search, and (LA)TeX can co-operate nicely.

5 Acknowledgements

Many thanks to Jerzy B. Ludwichowski, who has translated the abstract and keywords in Polish.

References

[1] *BaseX. Processing and Visualizing* XML *with a Native* XML *Database*. 2010. `http://www.inf.uni-konstanz.de/dbis/basex/index`.

[2] *GalaTex: an* XML *Full-Text Search Engine*. August 2005. `http://www.galaxquery.org/galatex/index.html`.

[3] Jean-Michel HUFFLEN: "XSLT 2.0 vs XSLT 1.0". In: *Proc. BachoTEX 2008 Conference*, pp. 67–77. April 2008.

[4] Jean-Michel HUFFLEN: "Introduction to XQuery". In: Tomasz PRZECHLEWSKI, Karl BERRY and Jerzy B. LUDWICHOWSKI, eds., *Proc. BachoTEX 2009 Conference*, pp. 17–25. April 2009.

[5] Jean-Michel HUFFLEN: "Processing 'Computed' Texts". *ArsTEXnica*, Vol. 8, pp. 102–109. In GUIT 2009 meeting. October 2009.

[6] Michael H. KAY: *XPath^{TM} 2.0 Programmer's Reference*. Wiley Publishing, Inc. 2004.

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

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

[9] W3C: XML *Path Language (XPath) 2.0*. W3C Recommendation Draft. Edited by Anders Berglund, Scott Boag, Don Chamberlin, Mary F. Fernández, Michael H. Kay, Jonathan Robie and Jérôme Siméon. January 2007. `http://www.w3.org/TR/2007/WD-xpath20-20070123`.

[10] W3C: *XQuery 1.0: an* XML *Query Language*. W3C Recommendation. Edited by Scott Boag, Don Chamberlin, Mary F. Fernández, Daniela Florescu, Jonathan Robie and Jérôme Siméon. January 2007. `http://www.w3.org/TR/xquery`.

[11] 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`.

[12] W3C: *XQuery Update Facility 1.0*. W3C Candidate Recommendation. Edited by Don Chamberlin, Daniela Florescu, Jim Melton, Jonathan Robie, and Jérôme Siméon. January 2008. `http://www.w3.org/TR/2008/CR-xquery-update-10-20080801/`.

[13] W3C: *XQuery and XPath Full Text 1.0*. W3C Candidate Recommendation. Edited by Sihem Amer-Yahia, Chavdar Botev, Stephen Buxton, Pat Case, Jochen Doerre, Mary Holstege, Jim Melton, Michael Rys, and Jayavel Shanmugasundaram. May 2008. `http://www.w3.org/TR/2008/CR-xpath-full-text-10-20080516/`.

[14] W3C: *XQuery 1.1*. W3C Working Draft. Edited by Don Chamberlin and Jonathan Siméon. December 2008. `http://www.w3.org/TR/xquery-11-20081203`.

[15] W3C: *XQuery and XPath Full Text 1.0*. W3C Candidate Recommendation. Edited by S. Amer-Yahia, Ch. Botev, S. Buxton, P. Case, J. Doerre, M. Holstege, J. Melton, M. Rys, and J. Shanmugasundaram. January 2010. `http://www.w3.org/TR/2008/CR-xpath-full-text-10-20100128/`.

[16] Priscilla WALMSLEY: *XQuery*. O'Reilly. April 2007.