Introduction to XQuery*

Jean-Michel HUFFLEN
LIFC (EA CNRS 4157)
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

XQuery is a query language for data stored in XML form. It 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). Like XSLT 2.0, it is based on XPath 2.0. We propose an introduction to XQuery, and some comparisons with XSLT allow readers to discern the applications XQuery is suitable for.
Keywords XML, XQuery (1.0 & 1.1), XPath 2.0, XSLT 2.0, generating (LA)TEX source texts.

Streszczenie

XQuery jest językiem zapytań dla danych przechowywanych w formacie XML. Może on być używany do wyszukiwania dokumentów w takim formacie albo w płaskich plikach tekstowych (potencjalnie użyteczne dla maszyny TEXowej). Podobnie jak XSLT 2.0, jest on oparty na XPath 2.0. Proponujemy wprowadzenie do XQuery oraz pewne porównanie z XSLT aby umożliwić czytelnikom podjęcie decyzji o tym, do jakich zastosowań XQuery jest przydatny.
Słowa kluczowe XML, XQuery (1.0 & 1.1), XPath 2.0, XSLT 2.0, generowanie tekstów źródłowych dla TEXa i LATEXa.

0 Introduction

This article continues a series of introductions to some languages and tools related to XML[1], presented at BachoTEX conferences [5, 6, 9]. As the language of transformations used for XML texts, XSLT[2] has already been presented, more precisely in its two versions: 1.0 [5, 6] and 2.0 [9]. In particular, we showed that XSLT was able to generate (LA)TEX source texts. Initially, XQuery was designed as a query language for collections of XML documents, as SQL[3] does for relational data bases. Even if our short introduction to this language does not aim to replace official documents [26, 31], we will show that this functional language is powerful and is able to build new XML or simple text documents, in particular (LA)TEX source texts.

XQuery is based on XPath, the language used to address parts of XML documents. More precisely, it uses XPath's new version (2.0) [24], like XSLT 2.0 [27]. Let us recall that every value handled within this data model is a *sequence*. An atomic value is a particular case of a sequence: a one-element sequence. For example, `(29,4,2009)` is a three-element sequence. Items being different types can be mixed in a sequence, e.g., `("Bachotek",2009,true())`. If a sequence is inserted into another one, the items of the inserted sequence become full-fledged items in the flattened resulting sequence, e.g.:

```
(("tu","Bachotek"),2009,("piwo","dobrze"))
```

is equivalent to:

```
("tu","Bachotek",2009,"piwo","dobrze")
```

XQuery's 'official' version is 1.0 [26] and a new one is in preparation, as a working draft presently [31]. As far as we know, zorba [34] is presently the only XQuery 1.1 processor. Our demonstrations will use saxon [13]. Other XQuery processors are Galax

* Title in Polish: *Wprowadzenie do XQuery*.

[1] e**X**tensible **M**arkup **L**anguage. Readers interested in a general introductory book to this formalism can refer to [18].

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

[3] **S**tructured **Q**uery **L**anguage. A good introductory book about it is [16].

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

<books>
  <omnibus series="Doc Savage">
    <author>     <!--  The organisation of author elements is the same than in [8].  -->
      <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>
    <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: File omnibuses.xml: specification of some stories collected in omnibus volumes.

[3] and AltovaXML [1], the latter is interesting if you work on a Windows operating system.

In the first section, we introduce XQuery's basic features. Section 2 makes precises XQuery's place within the world and tools of XML. Then we give a personal point of view in Section 3.

1 XQuery's features

As a progressive example's framework, let us consider the omnibuses.xml file given in Figure 1, already used in [9]. This XML text specifies the contents of some omnibus volumes; for each story included into such a book, we make precise its title and the year of its first publication.

1.1 Getting started

If we are searching the omnibuses.xml file given in Figure 1 and are looking for the title of the omnibus book containing the story entitled *Waves of Death*, we can use the following XPath expression[4]:

```
books/
omnibus[story/title = "Waves of Death"]/
booktitle
```

If saxon processes the following XQuery program:

```
doc("omnibuses.xml")/books/
omnibus[story/title = "Waves of Death"]/
booktitle
```

the result will look like:

```
<?xml version="1.0" encoding="UTF-8"?>
<booktitle>
  Doc Savage Omnibus #10
</booktitle>
```

Now let us assume the existence of a `$the-title` variable, bound to the string `"Waves of Death"`. Our query can become:

```
doc("omnibuses.xml")/books/
omnibus[story/title = $the-title]/booktitle
```

If we want the result to be stripped of the opening and closing tags, we use XPath 2.0's `data` function, returning the contents of a node-set's members [11, pp. 330–322]. In addition, the result is now surrounded by opening and closing `answer` tags, the curly braces force the embedded expression to be evaluated[5]:

```
<answer>{
```

[4] The following expression is suitable w.r.t. XPath 1.0 [22] and can be tested with the `xmllint` program [20], as shown in [5].

[5] This '`{...}`' notation generalises the *attribute value templates* used in XSLT [12, pp.116–119].

```
declare namespace xsd =
  "http://www.w3.org/2001/XMLSchema" ;
declare variable $the-title as xsd:string
  external ;

for $omnibus-0 as element(omnibus) in
      doc("omnibuses.xml")/books/omnibus,
    $title-0 as element(title) at $index in
      $omnibus-0/story/title
where $title-0 eq $the-title
return <answer index="{$index}">{
  data($omnibus-0/booktitle)
}</answer>
```

Figure 2: Searching omnibuses for a story's title.

```
  data(doc("omnibuses.xml")/books/
       omnibus[story/title = $the-title]/
       booktitle)
}</answer>
```

The same result can be reached by using a `for` expression that allows us to go along the node set of all the `omnibus` elements of this file. The `where` clause retains the `omnibus` elements such that the test yields `true`:

```
for $omnibus-0 in
      doc("omnibuses.xml")/books/omnibus
where $omnibus-0/story/title = $the-title
return <answer>{
  data($omnibus-0/booktitle)
}</answer>
```

We can nest two loop expressions: the outer one goes along all the `omnibus` elements, the inner one goes explores the successive `title` elements of each story[6]:

```
for $omnibus-0 in
      doc("omnibuses.xml")/books/omnibus,
    $title-0 in $omnibus-0/story/title
where $title-0 eq $the-title
return <answer>{
  data($omnibus-0/booktitle)
}</answer>
```

We can improve this search: the iteration number of the inner loop — starting at 1 — is now caught by means of a variable following the `at` keyword. We use this iteration number to give the rank — within the omnibus book — of the story we are looking for.

```
for $omnibus-0 in
      doc("omnibuses.xml")/books/omnibus,
    $title-0 at $index in
      $omnibus-0/story/title
where $title-0 eq $the-title
return <answer index="{$index}">{
  data($omnibus-0/booktitle)
}</answer>
```

The result looks like:

```
<?xml version="1.0" encoding="UTF-8"?>
<answer index="2">
  Doc Savage Omnibus #10
</answer>
```

Finally, the `$the-title` variable may be declared as follows:

```
declare variable $the-title :=
  "Waves of Death" ;
```

or declared as 'external' as we do in Figure 2. In this case, this variable's value *must* be supplied when the query is invoked[7], there is no default value for an external variable [26, § 2.1.2]. Figure 2 gives a definitive version of our first XQuery program. We add some type annotations [11, Ch. 9] to variables, by means of the `as` keyword. That allows a XQuery processor to perform type-checking. The type declaration for a general string is provided by XML

```
<items>
  <by-year year="1941">
    <title>The Invisible-Box Murders</title>
    <title>Birds of Death</title>
  </by-year>
  <by-year year="1942">
    <title>The Devil's Black Rock</title>
    <title>The Too-Wise Owl</title>
  </by-year>
  <by-year year="1943">
    <title>Waves of Death</title>
  </by-year>
  <by-year year="1945">
    <title>The Wee Ones</title>
    <title>Terror Takes 7</title>
    <title>Terror and the Lonely Widow</title>
  </by-year>
</items>
```

Figure 3: Grouping elements of Fig. 1's text.

[6] In addition, this reformulation allows the use of a *value comparison operator*. The '=' operator is used for *general sequence comparisons* and:

`$omnibus-0/story/title = $the-title`

reads 'is there an element common to the two sequences `$omnibus-0/story/title` and `$the-title`?' i.e., 'is there a `title` element whose contents is equal to `$the-title`'s value? The `eq` operator can be used only for atomic values such as strings, numbers, booleans, nodes. See [11, pp. 181–196] for more details.

[7] See how to do that with saxon in Appendix A.

```
declare namespace xsd =
  "http://www.w3.org/2001/XMLSchema" ;

declare variable $filename as xsd:string
  external ;

if (doc-available($filename)) then
  <items>{
    let $stories as element(story)* :=
      doc($filename)/books/omnibus/story
    for $year-e0 as xsd:untypedAtomic in
      distinct-values(data($stories/year))
    order by xsd:integer($year-e0)
    return <by-year year="{$year-e0}">{
      $stories[year eq $year-e0]/title
    }</by-year>
  }</items> else
  ()
```

Figure 4: Grouping stories' title by year in XQuery.

Schema's library[8] [23], so we use a prefix — '`xsd`' — to get access to this library's namespace[9] and show how to declare it.

1.2 A more ambitious example

The general expressions used in XQuery are FLWOR expressions. 'FLWOR' stands for '**F**or, **L**et, **W**here, **O**rder by, **R**eturn', the keywords used throughout such expressions. We already know the `for`, `where`, and `return` clauses. To discover the other clauses, let us consider again the XML text given in Figure 1 and group the titles of included stories by year, sorted increasingly. More precisely, we aim to get the XML text given in Figure 3.

The XQuery program of Figure 4, applied to the omnibuses.xml file, yields this result. A `let` clause binds variables to associated expressions without iteration, unlike a `for` clause. Practically, the variables of such a `let` clause are used to 'factorise' common subexpressions occurring in several places. An '`order by`' clause specifies how to sort the results of successive iterations. If this clause is absent, the global declaration:

```
declare ordering ordered ;
```

sorts such subresults according to the document order, whereas the declaration:

```
declare ordering unordered ;
```

[8] Schemas allow users to define *document types*, such a document type can be viewed as taxonomy common to some XML texts. There exist several schema languages, but only XML Schema [30] is interfaced with XSLT 2.0 and XQuery.

[9] Basic details about XML namespaces are given in [18, pp. 41–45].

```
xquery version "1.1" ;

declare namespace xsd =
  "http://www.w3.org/2001/XMLSchema" ;

declare variable $filename as xsd:string
  external ;

if (doc-available($filename)) then
  <items>{
    for $story-0 as element(story) in
      doc($filename)/books/omnibus/story
    let $year-0 as xsd:integer :=
      xsd:integer(data($story-0/year))
    group by $year-0
    order by $year-0
    return <by-year year="{$year-0}">{
      (: This is a comment. Here the following
         XPath expression concatenates all the titles
         of stories published the same year.
      :)
      $story-0/title
    }</by-year>
  }</items> else
  ()
```

Figure 5: Using a '`group by`' clause in XQuery 1.1.

leaves unspecified the order of appearance of results of a `for` clause's successive iterations. Such a declaration:

```
declare default order empty least ;
```

causes the empty sequence `()` to be ranked first — or last if '`greatest`' is put instead of '`least`'. Default conventions for such global declarations are implementation-dependent. Let us come back to '`order by`' clauses, here is a more complicated example:

```
stable order by last descending,
  first ascending empty least ;
```

which might be used to sort `personname` elements (cf. Fig. 1). It specifies a stable sort[10]; last names are sorted decreasingly, and person names left unsorted regarding last names are sorted increasingly[11] w.r.t. first names, an empty first name taking precedence over non-empty ones. XQuery allows the specification of precise language-dependent lexicographic orders for strings by means of *collations* [33, Ch. 17], identified by URIs[12]. The years of omnibus books' stories are sorted numerically since this '`order by`'

[10] A **stable** sort keeps the original order of items with equal sort keys.

[11] Of course, an order defaults to '`ascending`', but users can put this keyword down.

[12] **U**niform **R**esource **I**dentifier. The syntax is close to URLs' (**U**niform **R**esource **L**ocators) of the Web, but an URI

```
declare namespace saxon = "http://saxon.sf.net/" ;
declare namespace tu = "http://www.bachotex.pl/" ;
declare namespace xsd = "http://www.w3.org/2001/XMLSchema" ;

declare option saxon:output "omit-xml-declaration=yes" ;
(: Tells saxon to omit the processing declaration <?xml ...?> at the output's beginning. Such an order is
   implementation-dependent.
:)

declare variable $eol as xsd:string := "&#10;" ;
declare variable $filename as xsd:string external ;

declare function tu:escape-special-chars($y0 as element()) as xsd:string {
  replace(data($y0),"(#|%)","\\$1")
} ;

if (doc-available($filename)) then
  for $omnibus-0 as element(omnibus) in doc($filename)/books/omnibus return
      ("\item{$\bullet$}", tu:escape-special-chars($omnibus-0/booktitle), $eol,
       for $story-0 as element(story) in $omnibus-0/story return
           (" \itemitem{$\star$}", tu:escape-special-chars($story-0/title), $eol)),
  concat($eol, "\end", $eol) else
  ()
```

Figure 6: Generating a source text for Plain TEX.

clause uses information coerced into expressions of type `xsd:integer`[13] (cf. Fig. 4).

XQuery 1.1 [31] allows an additional '`group by`' clause partitioning an iteration's results. Let us compare Figures 4 & 5: in the first version, the iteration is done on the different years of publication and then we retain the `story` element whose `year` child element matches the current year; in the second case, all the `story` elements are grouped by year, and for each group, we put a `by-year` element surrounding the titles of all the elements of a group.

1.3 Deriving a *Plain TEX* source text

As shown in Figure 6, XQuery may be used to produce (LA)TEX source texts, even if it was not its initial purpose. When the successive atomic values of a sequence are displayed or copied onto an output file, they are separated by a space character. If you want to drop out this separator, just use XPath's `concat` function [11, pp. 312–313] as we did to build the last line: the `\end` command without preceding space character. Applying this XQuery program to the omnibuses.xml file results in the Plain TEX source text given in Figure 7. Two TEX's special characters — '`#`' and '`%`' — are escaped by a '\': we can see how to define a function to do that. XPath's `replace` function [11, pp. 400–403] uses regular expressions [11, Ch. 11] whose syntax originates from Perl[14].

1.4 More features

We have showed XQuery's basis. In addition, XQuery allows the creation of elements and attributes whose names are known at run-time [33, Ch. 5]. *Modules* allow definitions and queries to be reused in other contexts [33, Ch. 12]. Special processing based on the type of an expression is provided by `typeswitch` expressions [26, § 3.12.2]. XQuery 1.1 introduces *window iterations*, grouping consecutive items of an input sequence [31, § 3.8.4], and `try`/`catch` expressions, providing error handling [31, § 3.12], analogous to namesake statements in C++[15] [19].

In 'basic' XQuery, an expression never modifies the state of a document. XQuery Update Facility [28] is an extension that allows node insertions, modifications, or deletions. Another extension, XQuery and XPath Full-Text 1.0 [29], provides tools for full-text search, as well as structured search, against XML documents. Full-text search is different from substring search: the former searches for tokens and phrases rather than just substrings. Support for

only serves as a name and does not have to point to any resource [18, Ch. 3].

[13] The generic type `xsd:untypedAtomic` applies to all the atomic values that have no specific type [33, p. 438]. The results of XPath's `data` function [11, Ch. 10] are of `xsd:untypedAtomic` type.

[14] **P**ractical **E**xtraction **R**eport **L**anguage. A good introductory book to this language is [32].

[15] But there is no '`finally`' clause, as in Java [10] and C# [17].

```
\item{$\bullet$} Doc Savage Omnibus \#9
 \itemitem{$\star$} The Invisible-Box Murders
 \itemitem{$\star$} Birds of Death
 \itemitem{$\star$} The Wee Ones
 \itemitem{$\star$} Terror Takes 7
\item{$\bullet$} Doc Savage Omnibus \#10
 \itemitem{$\star$} The Devil's Black Rock
 \itemitem{$\star$} Waves of Death
 \itemitem{$\star$} The Too-Wise Owl
 \itemitem{$\star$} Terror and the Lonely Widow

\end
```

Figure 7: Applying Fig. 6's query to Fig. 1's text.

language-based search is provided, too. There is an implementation of this standard proposal: GalaTeX [2], built on top of the Galax processor [3].

2 XQuery within XML's world

At first glance, queries have the same look than XPath 2.0 expressions and XQuery programs seems to be able to perform document transformations, as XSLT does. The following subsections go throughly into these two points.

2.1 XQuery vs XPath

XPath 2.0 only provides 'simple' `for` expressions and does not allow end users to make precise variables' types[16], introducing index variables by means of the '`at`' keyword is not allowed, either:

```
for $story-0 in
      doc(...)/books/omnibus/story
return ...
```

It does not provide `let` expressions, the equivalent in 'pure' XPath 2.0 is a '`for`' expression along a one-element sequence:

```
for $story-0 in
      doc(...)/books/omnibus/story,
    $year-e0 in data($story-0/year)
return ...
```

A constant string is a valid XPath expression but the '`{...}`' notation does not belong to XPath.

2.2 XQuery vs XSLT

Here is what we can learn by reading and summarising W3C[17] documents. See [33, Ch. 25] for a 'more technical' view.

[16] But XSLT 2.0 allows variables and parameters of functions and templates to be typed by end users by means of the `as` attribute [12, pp. 73–76].

[17] **W**orld **W**ide **W**eb **C**onsortium.

The standards for XSLT 2.0 and XQuery were developed by separate working groups within W3C, operating together to ensure a common approach where appropriate. These two languages share the same data model, type system, and function library; both include XPath 2.0 as a sublanguage. However, they are rooted in different traditions and serve different communities' needs. XSLT was initially conceived as a stylesheet language whose primary goal is to render XML documents for human readers on screen, on the Web, or on paper. So, XSLT is stronger in its handling of narrative documents with more flexible structure, whilst XQuery is stronger in its data handling.

3 A personal synthesis

In the previous section, we tried to be as objective as possible; in this one, we give a personal point of view. The difference between XSLT and XQuery is close to the nuance between programming *by construction* and programming *with templates*. To illustrate that, let us consider the two following expressions in Scheme:

```
(let ((here 'Bachotek))
  (list 'staying 'at here 'Poland))
(let ((here 'Bachotek))
  `(staying at ,here Poland))
```

Both yield the same result:

```
(staying at Bachotek Poland)
```

In the first version, a function — `list` — is applied to arguments, possibly constant. In the second version, the "`` ` ``" character abbreviates a form that returns the following template, except when a comma appears, in which case the subexpression is evaluated and inserted at this place[18]. This second style is particularly useful when most of the desired structure is known in advance. The '`<...>{...}</...>`' notation may be viewed as programming with templates. So XQuery seems to us to be a good choice whenever this style is suitable. On the contrary, XSLT emphasises different processing w.r.t. elements and attributes of an input document.

This article is intentionally based on the same examples than [9]. So it can be noticed that XQuery programs are more compact and easier to understand. XQuery's main weakness is that many extensions are implementation-dependent, e.g., the use of character maps[19], interesting in XSLT 2.0 when we derive source texts for formats or engines built out of TEX, as shown in [9, § 6]. Using XPath's `replace` function is sufficient for simple texts, as we

[18] See [14, § 4.2.6] for more details.

[19] ... mentioned in [33, Table 22.1] about XQuery.

did in Figure 6, but may be more tedious if the input document contains non-alphanumerical characters, in particular, mathematical ones. In such a case, a solution might be to derive texts for a Unicode-compliant engine based on TEX[20], e.g., XETEX [15] or LuaTEX [4]. Another implementation-dependent feature is the separator between a sequence's consecutive items. In XSLT 2.0, this point is controlled by the `xsl:value-of` element's `separator` attribute [12, pp. 465–471]. In practice, the default rule — putting a space character — often suits us, as shown in Figures 6 & 7. Otherwise, it may result in many calls of XPath's `concat` function. To sum up about the derivation of source texts for TEX-based engines from XML documents, we think that XQuery is very suitable for such a task, except if the use of character maps is really needed.

4 Acknowledgements

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

A Giving a file name at run-time

As you can see whenever an XSLT processor is used, an XSLT stylesheet is applied to an XML document, the name of it being given at run-time[21]. For example, if you use the xsltproc processor [21], the command line that causes the grouping.xsl stylesheet [9, Fig. 3] to be applied to the omnibuses.xml file is:

```
xsltproc grouping.xsl omnibuses.xml
```

If this file does not exist, the error is signalled by the operating system in use. On the contrary, the name of the XML file processed by an XQuery request is hard-wired in most of examples given in official references [26, 30] or introductory books [33], as we did in Figure 2. In fact, there is no standard way to set it up outside XQuery programs [33, pp. 54 & 290]. Some implementations can start the execution of an XQuery program from a *context node*, so the request given in Figure 2 could be rewritten as:

```
for $omnibus-0 as ... in ./books/omnibus,
    ...
```

If saxon [13] is used, this new program, stored in a file named find-saxon.xq, can be invoked by setting the context node — given by the XPath expression '`.`' — to the root of an XML document by means of the '`-s:...`' option:

```
java net.sf.saxon.Query -s:omnibuses.xml \
  find-saxon.xq the-title="Waves of Death"
```

Similar methods can be used with AltovaXML [1] and Galax [3]. However, this *modus operandi* is not portable[22]. From our point of view, the best solution is to pass the file name by means of an external variable and return an empty sequence if the file is not processable. We can check that by means of the `doc-available` function [26, § 15.5.5]. So did we in Figures 2, 4 & 5.

B XQueryX

As shown by all the examples above, XQuery programs are not XML texts. XQuery's syntax is related to a 'classical' programming language's. XQueryX[23] [25] allows an XML representation of XQuery. The two syntaxes are merely different, but they express the same query semantics; in other words, the expressive power is the same. The main advantage for XQueryX texts is that they can be embedded directly in XML documents. On the contrary, XQuery's texts are more convenient for humans to read and write.

Figures 8 & 9 are a rewriting of our first example (cf. Fig. 2) using XQueryX's syntax. As you can see, the result, rooted by the `xqx:module` element, is very verbose[24] (!) but closely reflects XQuery statements' structure. We put this text only to give a representative idea about XQueryX's look. At the time of writing, there is no processor that directly deals with XQueryX texts. In fact, the normative document [25, App. B] includes an XSLT stylesheet converting XQueryX's syntax into 'classical' XQuery's syntax. Applying this stylesheet to the text of Figures 8 & 9 results in the program given in Figure 2.

References

[1] *AltovaXML*. 2008. `http://www.altova.com/altovaxml.html`.

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

[3] *Galax: the XQuery Implementation for Discriminating Hackers*. March 2009. `http://www.galaxquery.org`.

[20] Likewise, Polish texts built by means of XQuery should be processed by an engine accepting a suitable input encoding. The encoding of an XQuery program can be declared with the version used (cf. Fig. 5):

```
xquery version "1.0" encoding "ISO-8859-2" ;
```

[21] More exactly, an XSLT stylesheet is applied to a *main* document, other additional XML documents can be accessed by means of XPath's `doc` function [11, pp. 329–332].

[22] As a counter-example, another technique is used within Zorba [34].

[23] XQuery's **X**ML syntax.

[24] Some entities [18, pp. 45–53], introduced by means of a dummy `DOCTYPE` tag, as we did already in [7], allow us to enlighten this text.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE module [<!ENTITY child "<xqx:xpathAxis>child</xqx:xpathAxis>">
                 <!ENTITY xsd "<xqx:uri>http://www.w3.org/2001/XMLSchema</xqx:uri>">]>
<xqx:module xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xqx="http://www.w3.org/2005/XQueryX"
            xsi:schemaLocation="http://www.w3.org/2005/XQueryX http://www.w3.org/2005/XQueryX/xqueryx.xsd">
  <xqx:mainModule>
    <xqx:prolog>
      <xqx:namespaceDecl><xqx:prefix>xsd</xqx:prefix>&xsd;</xqx:namespaceDecl>
      <xqx:varDecl>
        <xqx:varName>the-title</xqx:varName>
        <xqx:typeDeclaration><xqx:atomicType xqx:prefix="xsd">string</xqx:atomicType></xqx:typeDeclaration>
        <xqx:external/>
      </xqx:varDecl>
    </xqx:prolog>
    <xqx:queryBody>
      <xqx:flworExpr>
        <xqx:forClause>
          <xqx:forClauseItem>
            <xqx:typedVariableBinding>
              <xqx:varName>omnibus-0</xqx:varName>
              <xqx:typeDeclaration>
                <xqx:elementTest>
                  <xqx:elementName><xqx:QName>omnibus</xqx:QName></xqx:elementName>
                </xqx:elementTest>
              </xqx:typeDeclaration>
            </xqx:typedVariableBinding>
            <xqx:forExpr>
              <xqx:pathExpr>
                <xqx:stepExpr>
                  <xqx:filterExpr>
                    <xqx:functionCallExpr>
                      <xqx:functionName>doc</xqx:functionName>
                      <xqx:arguments>
                        <xqx:stringConstantExpr><xqx:value>omnibuses.xml</xqx:value></xqx:stringConstantExpr>
                      </xqx:arguments>
                    </xqx:functionCallExpr>
                  </xqx:filterExpr>
                </xqx:stepExpr>
                <xqx:stepExpr>&child;<xqx:nameTest>books</xqx:nameTest></xqx:stepExpr>
                <xqx:stepExpr>&child;<xqx:nameTest>omnibus</xqx:nameTest></xqx:stepExpr>
              </xqx:pathExpr>
            </xqx:forExpr>
          </xqx:forClauseItem>
          <xqx:forClauseItem>
            <xqx:typedVariableBinding>
              <xqx:varName>title-0</xqx:varName>
              <xqx:typeDeclaration>
                <xqx:elementTest>
                  <xqx:elementName><xqx:QName>title</xqx:QName></xqx:elementName>
                </xqx:elementTest>
              </xqx:typeDeclaration>
            </xqx:typedVariableBinding>
            <xqx:positionalVariableBinding>index</xqx:positionalVariableBinding>
            <xqx:forExpr>
              <xqx:pathExpr>
                <xqx:stepExpr>
                  <xqx:filterExpr><xqx:varRef><xqx:name>omnibus-0</xqx:name></xqx:varRef></xqx:filterExpr>
                </xqx:stepExpr>
                <xqx:stepExpr>&child;<xqx:nameTest>story</xqx:nameTest></xqx:stepExpr>
                <xqx:stepExpr>&child;<xqx:nameTest>title</xqx:nameTest></xqx:stepExpr>
              </xqx:pathExpr>
            </xqx:forExpr>
```

Figure 8: Searching omnibuses for a story's title — version using XQueryX.

```
          </xqx:forClauseItem>
        </xqx:forClause>
        <xqx:whereClause>
          <xqx:eqOp>
            <xqx:firstOperand>
              <xqx:pathExpr>
                <xqx:stepExpr>
                  <xqx:filterExpr><xqx:varRef><xqx:name>title-0</xqx:name></xqx:varRef></xqx:filterExpr>
                </xqx:stepExpr>
              </xqx:pathExpr>
            </xqx:firstOperand>
            <xqx:secondOperand>
              <xqx:pathExpr>
                <xqx:stepExpr>
                  <xqx:filterExpr><xqx:varRef><xqx:name>the-title</xqx:name></xqx:varRef></xqx:filterExpr>
                </xqx:stepExpr>
              </xqx:pathExpr>
            </xqx:secondOperand>
          </xqx:eqOp>
        </xqx:whereClause>
        <xqx:returnClause>
          <xqx:elementConstructor>
            <xqx:tagName>answer</xqx:tagName>
            <xqx:attributeList>
              <xqx:attributeConstructor>
                <xqx:attributeName>index</xqx:attributeName>
                <xqx:attributeValueExpr>
                  <xqx:pathExpr>
                    <xqx:stepExpr>
                      <xqx:filterExpr><xqx:varRef><xqx:name>index</xqx:name></xqx:varRef></xqx:filterExpr>
                    </xqx:stepExpr>
                  </xqx:pathExpr>
                </xqx:attributeValueExpr>
              </xqx:attributeConstructor>
            </xqx:attributeList>
            <xqx:elementContent>
              <xqx:pathExpr>
                <xqx:stepExpr>
                  <xqx:filterExpr>
                    <xqx:functionCallExpr>
                      <xqx:functionName>data</xqx:functionName>
                      <xqx:arguments>
                        <xqx:pathExpr>
                          <xqx:stepExpr>
                            <xqx:filterExpr>
                              <xqx:varRef><xqx:name>omnibus-0</xqx:name></xqx:varRef>
                            </xqx:filterExpr>
                          </xqx:stepExpr>
                          <xqx:stepExpr>&child;<xqx:nameTest>booktitle</xqx:nameTest></xqx:stepExpr>
                        </xqx:pathExpr>
                      </xqx:arguments>
                    </xqx:functionCallExpr>
                  </xqx:filterExpr>
                </xqx:stepExpr>
              </xqx:pathExpr>
            </xqx:elementContent>
          </xqx:elementConstructor>
        </xqx:returnClause>
      </xqx:flworExpr>
    </xqx:queryBody>
  </xqx:mainModule>

</xqx:module>
```

Figure 9: Searching omnibuses for a story's title — version using XQueryX (Fig. 8 continued).

[4] Hans Hagen: "The Luafication of TEX and ConTEXt". In: *Proc. BachoTEX 2008 Conference*, pp. 114–123. April 2008.

[5] Jean-Michel Hufflen: "Introduction to xslt". *Biuletyn* gust, Vol. 22, pp. 64. In *BachoTEX 2005 conference*. April 2005.

[6] Jean-Michel Hufflen: "Advanced Techniques in xslt". *Biuletyn* gust, Vol. 23, pp. 69–75. In *BachoTEX 2006 conference*. April 2006.

[7] Jean-Michel Hufflen: "Introducing LATEX users to xsl-fo". tug*boat*, Vol. 29, no. 1, pp. 118–124. EuroBachoTEX 2007 proceedings. 2007.

[8] Jean-Michel Hufflen: "Revisiting Lexicographic Order Relations on Person Names". In: *Proc. BachoTEX 2008 Conference*, pp. 82–90. April 2008.

[9] Jean-Michel Hufflen: "xslt 2.0 vs xslt 1.0". In: *Proc. BachoTEX 2008 Conference*, pp. 67–77. April 2008.

[10] *Java Technology*. March 2008. http://java.sun.com.

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

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

[13] Michael H. Kay: *Saxon. The* xslt *and XQuery Processor*. March 2009. http://saxon.sourceforge.net.

[14] Richard Kelsey, William D. Clinger, Jonathan A. Rees, Harold Abelson, Norman I. Adams iv, David H. Bartley, Gary Brooks, R. Kent Dybvig, Daniel P. Friedman, Robert Halstead, Chris Hanson, Christopher T. Haynes, Eugene Edmund Kohlbecker, Jr, Donald Oxley, Kent M. Pitman, Guillermo J. Rozas, Guy Lewis Steele, Jr, Gerald Jay Sussman and Mitchell Wand: "Revised^5 Report on the Algorithmic Language Scheme". hosc, Vol. 11, no. 1, pp. 7–105. August 1998.

[15] Jonathan Kew: "XƎTEX in TEX Live and beyond". tug*boat*, Vol. 29, no. 1, pp. 146–150. EuroBachoTEX 2007 proceedings. 2007.

[16] Jim Melton and Alan R. Simon: *Understanding the new* sql. Morgan Kaufmann. 1993.

[17] Microsoft Corporation: *Microsoft C# Specifications*. Microsoft Press. 2001.

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

[19] Bjarne Stroustrup: *The C++ Programming Language*. 2nd edition. Addison-Wesley Publishing Company, Inc., Reading, Massachusetts. 1991.

[20] Daniel Veillard: *The* xml *C Parser and Toolkit of Gnome. libxml2*. http://xmlsoft.org. March 2003.

[21] Daniel Veillard: *The* xslt *C Library for Gnome*. http://xmlsoft.org/XSLT. March 2003.

[22] W3C: xml *Path Language (XPath). Version 1.0*. w3c Recommendation. Edited by James Clark and Steve DeRose. November 1999. http://www.w3.org/TR/1999/REC-xpath-19991116.

[23] W3C: xml *Schema Part 2: Datatypes*. w3c Recommendation. Edited by Paul V. Biron, Ashok Malhotra. October 2004. http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/.

[24] 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.

[25] W3C: xml *Syntax for XQuery 1.0 (XQueryX)*. w3c Recommendation. Edited by Jim Melton and Subramanian Muralidhar. January 2007. http://www.w3.org/TR/2007/REC-xqueryx-20070123.

[26] 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.

[27] 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.

[28] 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/.

[29] 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/.

[30] W3C: xml *Schema*. December 2008. http://www.w3.org/XML/Schema.

[31] W3C: *XQuery 1.1*. w3c Working Draft. Edited by Don Chamberlin and Jonathan Siméon. December 2008. http://www.w3.org/TR/xquery-11-20081203.

[32] Larry Wall, Tom Christiansen and Jon Orwant: *Programming Perl*. 3rd edition. O'Reilly & Associates, Inc. July 2000.

[33] Priscilla Walmsley: *XQuery*. O'Reilly. April 2007.

[34] *Zorba: the XQuery Processor*. 2009. http://www.zorba-xquery.com.