Xquery Lite 1.0

A query language for XML based on Xquery 1.0

Introduction

Status: This document describes Xquery-Lite 1.0 new features matching closer the Xquery 1.0 spec will be added to future versions of this language. New versions will always be compatible with older versions. This language may be extended but won't be changed.

Xquery Lite is a subset of the W3C's Xquery 1.0 language. Xquery is a powerful language for querying XML documents. We simplified the Xquery spec obtaining a sub-set that we called Xquery Lite that is easy to implement and allows many very interesting features for XML-Querying.

Characteristics

Xquery is based on Xpath so Xquery Lite is also based on Xpath you can think about Xquery Lite as a super-set of Xpath adding many features useful for querying XML documents, the most important feature is that queries are not only restricted to one XML document, you can "join" many documents and output the result comparing elements in two different documents if you want.

Xquery Lite 1.1 preview

In a future 1.1 version of the language we plan to add the following features:

- IF statements
- A sequence-equal function that compares subtrees regardless of the order in which elements appear
- A "some" function allowing constructs in the form where some($b/title) = "foo" thus comparing some element in a node set agains another element or value.
- Sorting
- Min, Max and Average functions

An Xquery Lite 1.0 query

An Xquery Lite query is composed by decorating text and Xquery Lite expressions, expressions must be enclosed between {brackets}, everything outside brackets is directly outputted to the result

Example of a Query skeleton:

```
<result>
{
 XQUERY_EXPRESSIONS
}
</result>
```

Note that the decorating text doesn't need to be XML it can be plain text or whatever you want for the result of the query.

Xquery Lite expressions

Xquery Lite expressions are based on Xquery FLWR (For-Let-Where-Return) expressions. A FLWR expression is in the form:

```
FOR_STATEMENT+
LET_STATEMENT*
WHERE_STATEMENT?
RETURN_STATEMENT
```

So we must have at least a FOR statement, followed by 0..n LET statements, then an optional where statement and only one mandatory return statement. As we can see later a return statement can include other queries thus allowing the use of nested-queries or sub-queries. You don't need to sepparate FOR-LET-WHERE-RETURN by line breaks but you do must leave at least one empty whitespace (tab,blank or newline) character between statements.

FLWR-Lite tutorial

We are going to explain now the meaning of a FLWR-lite expression (An Xquery-Lite expression).

FOR statementss

A FOR statement executes an Xpath expression over an XML document returning a nodeset, then remaining sentences in the expression will be evaluated once for each node in the nodeset.

Example:

```
{
  for $a in document("foo1.xml")//name
  return {$a}
}
```

The above expression will get all the "name" elements from the XML

document contained in the file "foo1.xml" and for each "name" element the return expression will be executed returning the "name" element. The result will be something like:

```
<name>foo</name><name>John</name>...<name>Peter</name>
```

As you can see each node will be "binded" to the $a variable ($a is not a PHP variable, is a query variable) and we may use $a in statements below the for statement to consult the value of the current node

So we know what a FOR statement does, let's see how many ways do we have for a fot statement

```
for $a in
document("foo1.xml")/xpath_expression    to
query a document from a file
for $a in xmlmem($xml)/xpath_expression
to query a document from a php string
for $a in $x/xpath_expression
to query a document from a Xquery Lite
variable
```

So we can execute Xpath expressions on files, strings or a query variable binded by a previous for or let statement

When two or more fors are used instructions below the "fors" will be executed for each combination of elements generated by the nodesets returned by each for

Example:

```
<result>
{
  for $a in document("foo1.xml")//name
  for $b in document("people.xml")//item
  return <combination>
         {$a}
         {$b}
         </combination>
}
</result>
```

This will generate something like:

```
<result>
  <combination>
    <name>foo</name>
    <item>i1</item>
  </combination>
```

```
  <combination>
    <name>foo</name>
    <item>i2</item>
  </combination>
  ...
</result>
```

So two or more for statements produce a "join" of XML documents.

Two or more for statements can be written in a compact syntax using comma sepparated for expressions:

```
for $a in document("foo1.xml")//item
for $t in $a/title
for $n in $a/name
```

Can be written as:

```
for $a in document("foo1.xml")//item, $t in
$a/title, $n in $a/name
```

Return statement

A return statement indicates what the Query must return, returns are executed once for each combination of nodes produced by for statements, a return statement has the following syntax.

```
return Xquery-Lite-Query
```

So what follows a return statement can be text decorations and Xquery expressions enclosed between brackets, typically text decoration is used to give some format to the answer and Xquery expressions are used to take values from query variables and output them.

Example:

```
{
 for $a in document("foo1.xml")//name
 return <a_name>
          {$a/text()}
        </a_name>
}
```

Note that the for statements produce name "elements" so if we want to return the text we use a mini-xpath expression using each "name" element as the XML source inside the return statement. We may have several Xquery expressions in a return expression enclosed by brackets and each of them can be a full FLWR expression so we may have sub-queries using query variables as XML sources and producing some result.

Where statement

A where statement is used to "filter" results that will be passed to the return statement, it is the "key" construction for a query since it allows us to return what we want and not everything. A where statement must return true or false, for example:

```
{
 for $a in document("foo1.xml")//person
 where $a/age > 10
 return
          {$a/name/text()}

}
```

In this example we return all the names of persons with age > 10 note that an age "element" when compared to a string or an integer is automatically converted to a string.

In a where expression we may have:

- Parenthesis can be used to override precedence and are recommended.
- Expressions can be combined by "and" and "or" expressions ex: where ($a/name="foo") and ($a/age>10)<60
- <,>,<=,>=,=,<> can be used to compare elements against values or elements agains elements
- If an element with sub-elements is compared to another elements a recursive-deep match is done, if all the subelements are present and equal then the elements are considered equal.
- +,-,*,/ can be used to perform arithmetic operations between elements containing numbers
- The count() function can be used to count the number of node in a expression ex: where count($a/name)=1 if we want elements with only one name subelement

So some complex where expressions can be used to perform complex queries.

Let statements

A let statement can bind a value to a query variable, for example:

```
let $name := $a/name/text()
```

This expression binds the result of the $a/name/text() xpath expression to the query's $name variable.

Let expressions are generally optional and used only as auxiliary variables

that will later be used in a where or return expression

Examples of Xquery-Lite queries

Now we are going to show how some W3C Xquery Use-Cases can be solved using Xquery Lite

In the examples we will work with the "bib.xml" document which has information about books:

```
<bib>
    <book year="1994">
        <title>TCP/IP Illustrated</title>

<author><last>Stevens</last><first>W.</first></author>
        <publisher>Addison-Wesley</publisher>
        <price> 65.95</price>
    </book>

    <book year="1992">
        <title>Advanced XML Programming in the Unix
environment</title>

<author><last>Stevens</last><first>W.</first></author>
        <publisher>Addison-Wesley</publisher>
        <price>65.95</price>
    </book>

    <book year="2000">
        <title>Data on the Web</title>

<author><last>Abiteboul</last><first>Serge</first></author>

<author><last>Buneman</last><first>Peter</first></author>

<author><last>Suciu</last><first>Dan</first></author>
        <publisher>Morgan Kaufmann Publishers</publisher>
        <price> 39.95</price>
    </book>

    <book year="1999">
        <title>The Economics of Technology and Content for
Digital TV</title>
        <editor>
               <last>Gerbarg</last><first>Darcy</first>
                <affiliation>CITI</affiliation>
        </editor>
            <publisher>Kluwer Academic
Publishers</publisher>
        <price>129.95</price>
    </book>

</bib>
```

And the reviews.xml document listing book reviews:

```
<reviews>
    <entry>
        <title>Data on the Web</title>
        <price>34.95</price>
        <review>
               A very good discussion of
semi-structured database
               systems and XML.
        </review>
    </entry>
    <entry>
        <title>Advanced Programming in the
Unix environment</title>
        <price>65.95</price>
        <review>
               A clear and detailed
discussion of UNIX programming.
        </review>
    </entry>
    <entry>
        <title>TCP/IP Illustrated</title>
        <price>65.95</price>
        <review>
               One of the best books on
TCP/IP.
        </review>
    </entry>
</reviews>
```

Use-Case 1

Query: List books published by Addison-Wesley after 1991, including their year and title.

Solution:

```
<bib>
 {
  for $b in
document("c:\apache\htdocs\phpxmlclasses\bib.xml")/bib/book
  where $b/publisher = "Addison-Wesley" and $b/@year > 1991
  return
    <book year="{ $b/@year }">
     { $b/title }
    </book>
 }
</bib>
```

Comment: we retrieve all book elements using a for statement and filter by the name of the published and the year attribute (note the element/@name

notation). The return just builds an XML document listing the year as an attribute and the title of the books that match the criteria.

The result will be something like this:

```
<bib>
<book year="1994">
<title>TCP/IP Illustrated</title>
</book>
<book year="1992">
<title>Advanced XML Programming in the Unix
environment</title>
</book>
</bib>
```

Use-Case 2

Query: Create a flat list of all the title-author pairs, with each pair enclosed in a "result" element.

```
<results>
  {
    for $b in
document("c:\apache\htdocs\phpxmlclasses\bib.xml")/bib/book,$t
in $b/title,$a in $b/author
    return
        <result>
            { $t }
            { $a }
        </result>
  }
</results>
```

Comment: We retrieve all the book elements then for each book we retrieve all the titles (a book may have several titles) and all the book's authors and generate a result element with a title sub-element and author sub-element.

Expected result:

```
<results>
  <result>
    <title>TCP/IP Illustrated</title>
    <author><last>Stevens</last><first>W.</first></author>
  </result>
  <result>
    <title>Advanced XML Programming in the Unix
environment</title>
    <author><last>Stevens</last><first>W.</first></author>
  </result>
  <result>
    <title>Data on the Web</title>
```

```
<author><last>Abiteboul</last><first>Serge</first></author>
  </result>
  <result>
    <title>Data on the Web</title>

<author><last>Buneman</last><first>Peter</first></author>
  </result>
  <result>
    <title>Data on the Web</title>
    <author><last>Suciu</last><first>Dan</first></author>
  </result>
</results>
```

Use-Case 3

Query: For each book in the bibliography, list the title and authors, grouped inside a "result" element.

Solution:

```
<results>
{
    for $b in
document("c:\apache\htdocs\phpxmlclasses\bib.xml")/bib/book
    return
        <result>
            { $b/title }
            { $b/author  }
        </result>
}
</results>
```

Comment: In this simple query we just return the title and author for each book note that since a book may have many authors we may have several author elements for each result.

Expected result:

```
<results>
 <result>
   <title>TCP/IP Illustrated</title>
   <author><last>Stevens</last><first>W.</first></author>
 </result>
 <result>
   <title>Advanced XML Programming in the Unix
environment</title>
   <author><last>Stevens</last><first>W.</first></author>
 </result>
 <result>
   <title>Data on the Web</title>
```

```
<author><last>Abiteboul</last><first>Serge</first></author>

<author><last>Buneman</last><first>Peter</first></author>
   <author><last>Suciu</last><first>Dan</first></author>
 </result>
 <result>
 <title>The Economics of Technology and Content for Digital
TV</title>
 </result>
</results>
```

Use-Case 4

For each author in the bibliography, list the author's name and the titles of all books by that author, grouped inside a "result" element.

Solution:

```
<results>
  {
    for $a in distinct-
values(document("c:\apache\htdocs\phpxmlclasses\bib.xml")//author)
    return
        <result>
            {$a }
            {
                for $b in
document("c:\apache\htdocs\phpxmlclasses\bib.xml")/bib/book
                for $a2 in $b/author
                where $a2=$a
                return {$b/title}
            }
        </result>
  }
</results>
```

Comment: We retrieve all the authors using a for statement then for each author we return something so we use a return statement with sub-queries. First we output the author element and then for each book in bib where the author is our author we output the title. This way we output all the titles by this author. Note the distict-values function that can be used a a wrapper of an expression to filter duplicate elements in a nodeset.

Expected result:

```
<results>
 <result>
   <author><last>Stevens</last><first>W.</first></author>
   <title>TCP/IP Illustrated</title>
   <title>Advanced XML Programming in the Unix
```

```
environment</title>
 </result>
 <result>

<author><last>Abiteboul</last><first>Serge</first></author>
   <title>Data on the Web</title>
 </result>
 <result>

<author><last>Buneman</last><first>Peter</first></author>
   <title>Data on the Web</title>
 </result>
 <result>
   <author><last>Suciu</last><first>Dan</first></author>
   <title>Data on the Web</title>
 </result>
</results>
```

Use-Case 5

For each book found at both bn.com and amazon.com, list the title of the book and its price from each source.

Solution:

```
<books-with-prices>
  {
    for $b in
document("c:\apache\htdocs\phpxmlclasses\bib.xml")//book,
        $a in
document("c:\apache\htdocs\phpxmlclasses\reviews.xml")//entry
    where $b/title = $a/title
    return
        <book-with-prices>
            { $b/title }
            <price-amazon>{ $a/price/text() }</price-amazon>
            <price-bn>{ $b/price/text() }</price-bn>
        </book-with-prices>
  }
</books-with-prices>
```

Comment: This is avery nice example since two XML documents are "joined". We retrieve all the books from bib.xml and all the reviews from reviews.xml and we filter matching the titles (the join condition). Then we just list the price a book has in one document and the other one. Easy to think, easy to write and the result is just what we wanted!

Expected result:

```
<books-with-prices>
 <book-with-prices>
```

```
   <title>TCP/IP Illustrated</title>
   <price-amazon>65.95</price-amazon>
   <price-bn> 65.95</price-bn>
 </book-with-prices>
 <book-with-prices>
   <title>Data on the Web</title>
   <price-amazon>34.95</price-amazon>
   <price-bn> 39.95</price-bn>
 </book-with-prices>
</books-with-prices>
```

Use-Case 6

For each book that has at least one author, list the title and first two authors, and an empty "et-al" element if the book has additional authors.

Solution:

```
  {
    for $b in xmlmem($bib)//book
    where count($b/author) > 2
    return

            { $b/title }
            {
                for $a in
$b/author[position()<=2]
                return {$a}
            }


  }

  {
    for $b in xmlmem($bib)//book
    where count($b/author) <= 2
    return

            { $b/title }
            {
                for $a in
$b/author[position()<=2]
                return {$a}
            }

  }
```

Comment: The Xquery Lite 1.0 language doesn't have an IF statement yet (it is planned for 1.1) so we have to split the query in two parts one for the books with more than two authors and another part for the books with 2 or

less authors. Note that the books document has to be parse twice which is not good. This will be solved when the if statement is ready meanwhile we cansee that the use-case can be solved anyway.

Expected result:

```
<bib>
 <book>
   <title>Data on the Web</title>

<author><last>Abiteboul</last><first>Serge</first></author>

<author><last>Buneman</last><first>Peter</first></author>
   <et-al />
 </book>
 <book>
   <title>TCP/IP Illustrated</title>
   <author><last>Stevens</last><first>W.</first></author>
 </book>
 <book>
   <title>Advanced XML Programming in the Unix
environment</title>
   <author><last>Stevens</last><first>W.</first></author>
 </book>
 <book>
   <title>The Economics of Technology and Content for
Digital TV</title>
  </book>
</bib>
```

Use-Case 9

In the document "bib.xml", find all titles that contain the word "XML", regardless of the level of nesting.

Solution:

```
<results>
{
    for $t in
document("c:\apache\htdocs\phpxmlclasses\bib.xml")//title[contains(./text(),"XML")]
    return {$t}
  }
</results>
```

Comment: A straightforward application of an Xpath query. Note how easy it is to enhance Xpath queries giving the result an XML format using simple Xquery Lite queries.

Expected result:

```
<results>
 <title>Advanced XML Programming in the
Unix environment</title>
</results>
```

Use-Case 11

For each book with an author, return the book with its title and authors. For each book with an editor, return a reference with the book title and the editor's affiliation.

Solution:

```
<bib>
{
        for $b in
document("c:\apache\htdocs\phpxmlclasses\bib.xml")//book[author]
        return
            <book>
                { $b/title }
                { $b/author }
            </book>
}
{
        for $b in
document("c:\apache\htdocs\phpxmlclasses\bib.xml")//book[editor]
        return
          <reference>
            { $b/title }
            {$b/editor/affiliation}
          </reference>
}
```

Comment: This example shows how to combine two Xquery Lite expressions in a query note the Xpath expression used to check if the book element has an author or editor subelement.

Expected result:

```
<bib>
 <book>
   <title>TCP/IP Illustrated</title>
   <author><last>Stevens</last><first>W.</first></author>
 </book>
 <book>
   <title>Advanced XML Programming in the Unix
environment</title>
   <author><last>Stevens</last><first>W.</first></author>
 </book>
```

```
 <book>
   <title>Data on the Web</title>

<author><last>Abiteboul</last><first>Serge</first></author>

<author><last>Buneman</last><first>Peter</first></author>
   <author><last>Suciu</last><first>Dan</first></author>
 </book>

 <reference>
   <title>The Economics of Technology and Content for
Digital TV</title>
   <affiliation>CITI</affiliation>
 </reference>
</bib>
```

Implementations

There's only one implementation of the Xquery Lite language by the moment:

- PHP: Class Xquery Lite a full Xquery Lite 1.0 implementation for PHP based on the DOM standard. This implementation has a demo site at this site