XML Syntax for XQuery 1.0 (XQueryX)

W3C Working Draft 07 June 2001

This version:
http://www.w3.org/TR/2001/WD-xqueryx-20010607
Latest version:
http://www.w3.org/TR/xqueryx
Editors:
Ashok Malhotra (Microsoft) <ashokma@microsoft.com>
Jonathan Robie (Software AG) <jonathan.robie@softwareagusa.com>
Michael Rys (Microsoft) <mrys@microsoft.com>

Abstract

This paper presents an XML Syntax for [XQuery Working Draft].

Status of this document

This document is the first publicly available W3C Working Draft of XQueryX, for review by W3C members and other interested parties. It is a draft document and may be updated, replaced, or made obsolete by other documents at any time. The XQueryX DTD and XML Schema will track the XQuery 1.0 syntax and will be changed as often as the XQuery 1.0 syntax is changed in future Working Drafts. It is inappropriate to use W3C Working Drafts as reference material or to cite them as other than "work in progress".

Comments on this document should be sent to the W3C mailing list www-xml-query-comments@w3.org, which is archived at http://lists.w3.org/Archives/Public/www-xml-query-comments/.

This document was produced by the W3C XML Query Working Group, which is part of the W3C XML Activity. A list of current W3C Recommendations and other technical documents can be found at http://www.w3.org/TR/.

Table of contents

Appendices

---

1 Introduction

The [XML Query 1.0 Requirements] states that "The XML Query Language MAY have more than one syntax binding. One query language syntax MUST be convenient for humans to read and write. One query language syntax MUST be expressed in XML in a way that reflects the underlying structure of the query."

XQueryX is an XML representation of an XQuery. It was created by mapping the productions of the XQuery abstract syntax directly into XML productions. The result is not particularly convenient for humans to read and write, but it is easy for programs to parse, and because XQueryX is represented in XML, standard XML tools can be used to create, interpret, or modify queries.

There are several environments in which XQueryX may be useful:

- *Parser Reuse.* In heterogeneous data environments, a variety of systems may be used to execute a query. One parser can generate XQueryX for all of these systems.
- *Queries on Queries.* Because XQueryX is represented in XML, queries can be queried and can be transformed into new queries. For instance, a query can be performed against a set of XQueryX queries to determine which queries use FLWR expressions to range over a set of invoices.
- *Generating Queries.* In some XML-oriented programming environments, it may be more convenient to build a query in its XQueryX representation than in the corresponding XQuery representation, since XML tools can be used to do so.
- *Embedding Queries in XML.* XQueryX can be embedded directly in an XML document.

2 Mapping the Syntax

XQueryX is a close representation of the abstract syntax found in Appendix B of the [XQuery Working Draft]. For each production in the abstract syntax, we created an equivalent XML representation. For instance, the abstract syntax contains the following productions:

```
FLWRExpr    ::=    (ForClause | LetClause)+ WhereClause? "return" Expr
ForClause    ::=    "for" Variable "in" Expr ("," Variable "in" Expr)*
LetClause    ::=    "let" Variable ":=" Expr ("," Variable ":=" Expr)*
WhereClause    ::=    "where" Expr
```

The following XQueryX content models closely mirror the structure of the above productions:

```
<!ELEMENT flwr ((forAssignment | letAssignmen)+, where?, return)>
<!ELEMENT forAssignment %expression;>
      <!ATTLIST forAssignment variable CDATA #REQUIRED>
<!ELEMENT letAssignment %expression; >
      <!ATTLIST letAssignment variable CDATA #REQUIRED >
<!ELEMENT where (%expression;)>
<!ELEMENT return (%expression;)>
```

Since XQuery uses the Expression production liberally to allow expressions to be flexibly combined, XQueryX uses the %expression parameter entity in these same contexts to allow all expression types to occur.

Now consider a FLWR expression in XQuery:

```
FOR $b IN document("bib.xml")//book
WHERE $b/publisher = "Morgan Kaufmann" AND $b/year = "1998"
RETURN
    $b/title
```

The equivalent in XQueryX is as follows:

```
<q:query xmlns:q="http://www.w3.org/2001/06/xqueryx">
 <q:flwr>
   <q:forAssignment variable="$b">
     <q:step axis="SLASHSLASH">
       <q:function name="document">
         <q:constant datatype="CHARSTRING">bib.xml</q:constant>
       </q:function>
       <q:identifier>book</q:identifier>
     </q:step>
   </q:forAssignment>
   <q:where>
     <q:function name="AND">
       <q:function name="EQUALS">
         <q:step axis="CHILD">
           <q:variable>$b</q:variable>
           <q:identifier>publisher</q:identifier>
         </q:step>
         <q:constant datatype="CHARSTRING">Morgan Kaufmann</q:constant>
       </q:function>
       <q:function name="EQUALS">
         <q:step axis="CHILD">
           <q:variable>$b</q:variable>
           <q:identifier>year</q:identifier>
         </q:step>
         <q:constant datatype="CHARSTRING">1998</q:constant>
       </q:function>
     </q:function>
   </q:where>
   <q:return>
     <q:step axis="CHILD">
       <q:variable>$b</q:variable>
       <q:identifier>title</q:identifier>
     </q:step>
   </q:return>
```

```
  </q:flwr>
</q:query>
```

Note that path expressions are expanded to show their structure. Also, note that the prefix syntax for binary operators like AND makes the precedence explicit. In general, humans find it easier to read an XML representation that does not expand path expressions, but it is less convenient for programmatic representation. We are not proposing XQueryX as a convenient syntax for humans to read and write, so we slant our representation toward the programmer.

The appendices of this document provide an XML Schema and a DTD that define the entire XQueryX language. In the rest of this paper, we will show the XQueryX generated for several queries.

3 Examples from the XQuery Working Draft in XML Syntax

3.1 Example 1

Here is Q13 from the the [XQuery Working Draft] : "List each publisher and the average price of its books."

```
FOR $p IN distinct(document("bib.xml")//publisher)
LET $a := avg(document("bib.xml")//book[publisher = $p]/price)
RETURN
    <publisher>
        <name>{ $p/text() }</name>
        <avgprice>{ $a }</avgprice>
    </publisher>
```

Here is the equivalent XML syntax.

```
<q:query xmlns:q="http://www.w3.org/2001/06/xqueryx">
  <q:flwr>
    <q:forAssignment variable="$p">
      <q:function name="distinct">
        <q:step axis="SLASHSLASH">
          <q:function name="document">
            <q:constant datatype="CHARSTRING">bib.xml</q:constant>
          </q:function>
          <q:identifier>publisher</q:identifier>
        </q:step>
      </q:function>
    </q:forAssignment>
    <q:letAssignment variable="$a">
      <q:function name="avg">
        <q:step axis="CHILD">
          <q:function name="document">
            <q:constant datatype="CHARSTRING">bib.xml</q:constant>
          </q:function>
          <q:step axis="CHILD">
            <q:predicatedExpr>
              <q:identifier>book</q:identifier>
              <q:predicate>
                <q:function name="EQUALS">
                  <q:identifier>publisher</q:identifier>
                  <q:variable>$p</q:variable>
                </q:function>
              </q:predicate>
            </q:predicatedExpr>
```

```
            <q:identifier>price</q:identifier>
          </q:step>
        </q:step>
      </q:function>
    </q:letAssignment>
    <q:return>
      <q:elementConstructor>
        <q:tagName>
          <q:identifier>publisher</q:identifier>
        </q:tagName>
        <q:elementConstructor>
          <q:tagName>
            <q:identifier>name</q:identifier>
          </q:tagName>
          <q:step axis="CHILD">
            <q:variable>$p</q:variable>
            <q:nodeKindTest kind="TEXT" />
          </q:step>
        </q:elementConstructor>
        <q:elementConstructor>
          <q:tagName>
            <q:identifier>avgprice</q:identifier>
          </q:tagName>
          <q:variable>$a</q:variable>
        </q:elementConstructor>
      </q:elementConstructor>
    </q:return>
  </q:flwr>
</q:query>
```

Note the representation of a function call. Since avg() is a built-in function, the function declaration is not represented in the above query.

3.2 Example 2

Here is Q15 from the the [XQuery Working Draft] : "Invert the structure of the input document so that, instead of each book element containing a list of authors, each distinct author element contains a list of book-titles."

```
<author_list>
  {
    FOR $a IN distinct(document("bib.xml")//author)
    RETURN
        <author>
            <name>{ $a/text() }</name>
               {
                FOR $b IN document("bib.xml")//book[author = $a]
                RETURN $b/title
               }
        </author>
  }
</author_list>
```

This can be represented in XML syntax as

```
<q:query xmlns:q="http://www.w3.org/2001/06/xqueryx">
  <q:elementConstructor>
    <q:tagName>
```

```
        <q:identifier>author_list</q:identifier>
      </q:tagName>
      <q:flwr>
        <q:forAssignment variable="$a">
          <q:function name="distinct">
            <q:step axis="SLASHSLASH">
              <q:function name="document">
                <q:constant datatype="CHARSTRING">bib.xml</q:constant>
              </q:function>
              <q:identifier>author</q:identifier>
            </q:step>
          </q:function>
        </q:forAssignment>
        <q:return>
          <q:elementConstructor>
            <q:tagName>
              <q:identifier>author</q:identifier>
            </q:tagName>
            <q:elementConstructor>
              <q:tagName>
                <q:identifier>name</q:identifier>
              </q:tagName>
              <q:step axis="CHILD">
                <q:variable>$a</q:variable>
                <q:nodeKindTest kind="TEXT" />
              </q:step>
            </q:elementConstructor>
            <q:flwr>
              <q:forAssignment variable="$b">
                <q:step axis="SLASHSLASH">
                  <q:function name="document">
                    <q:constant datatype="CHARSTRING">bib.xml</q:constant>
                  </q:function>
                  <q:predicatedExpr>
                    <q:identifier>book</q:identifier>
                    <q:predicate>
                      <q:function name="EQUALS">
                        <q:identifier>author</q:identifier>
                        <q:variable>$a</q:variable>
                      </q:function>
                    </q:predicate>
                  </q:predicatedExpr>
                </q:step>
              </q:forAssignment>
              <q:return>
                <q:step axis="CHILD">
                  <q:variable>$b</q:variable>
                  <q:identifier>title</q:identifier>
                </q:step>
              </q:return>
            </q:flwr>
          </q:elementConstructor>
        </q:return>
      </q:flwr>
    </q:elementConstructor>
  </q:query>
```

3.3 Example 3

Here is Q19 from the the [XQuery Working Draft] : " Make an alphabetic list of publishers. Within each

publisher, make a list of books, each containing a title and a price, in descending order by price." It illustrates the structure of sorting expressions in XQueryX.

```
<publisher_list>{
    FOR $p IN distinct(document("bib.xml")//publisher)
    RETURN
        <publisher>
            <name>{ $p/text() }</name>
            {
                FOR $b IN document("bib.xml")//book[publisher = $p]
                RETURN
                    <book>
                        { $b/title }
                        { $b/price }
                    </book>
                SORTBY(price DESCENDING)
            }
        </publisher>
    SORTBY(name)
}</publisher_list>
```

Here is the equivalent XML syntax.

```
<q:query xmlns:q="http://www.w3.org/2001/06/xqueryx">
  <q:elementConstructor>
    <q:tagName>
      <q:identifier>publisher_list</q:identifier>
    </q:tagName>
    <q:sortBy>
      <q:flwr>
        <q:forAssignment variable="$p">
          <q:function name="distinct">
            <q:step axis="SLASHSLASH">
              <q:function name="document">
                <q:constant datatype="CHARSTRING">bib.xml</q:constant>
              </q:function>
              <q:identifier>publisher</q:identifier>
            </q:step>
          </q:function>
        </q:forAssignment>
        <q:return>
          <q:elementConstructor>
            <q:tagName>
              <q:identifier>publisher</q:identifier>
            </q:tagName>
            <q:elementConstructor>
              <q:tagName>
                <q:identifier>name</q:identifier>
              </q:tagName>
              <q:step axis="CHILD">
                <q:variable>$p</q:variable>
                <q:nodeKindTest kind="TEXT" />
              </q:step>
            </q:elementConstructor>
            <q:sortBy>
              <q:flwr>
                <q:forAssignment variable="$b">
                  <q:step axis="SLASHSLASH">
                    <q:function name="document">
```

```
                        <q:constant datatype="CHARSTRING">bib.xml</q:constant>
                      </q:function>
                      <q:predicatedExpr>
                        <q:identifier>book</q:identifier>
                        <q:predicate>
                          <q:function name="EQUALS">
                            <q:identifier>publisher</q:identifier>
                            <q:variable>$p</q:variable>
                          </q:function>
                        </q:predicate>
                      </q:predicatedExpr>
                    </q:step>
                  </q:forAssignment>
                  <q:return>
                    <q:elementConstructor>
                      <q:tagName>
                        <q:identifier>book</q:identifier>
                      </q:tagName>
                      <q:step axis="CHILD">
                        <q:variable>$b</q:variable>
                        <q:identifier>title</q:identifier>
                      </q:step>
                      <q:step axis="CHILD">
                        <q:variable>$b</q:variable>
                        <q:identifier>price</q:identifier>
                      </q:step>
                    </q:elementConstructor>
                  </q:return>
                </q:flwr>
                <q:sortfield order="DESCENDING">
                  <q:identifier>price</q:identifier>
                </q:sortfield>
              </q:sortBy>
            </q:elementConstructor>
          </q:return>
        </q:flwr>
        <q:sortfield>
          <q:identifier>name</q:identifier>
        </q:sortfield>
      </q:sortBy>
    </q:elementConstructor>
  </q:query>
```

Note that the XQueryX representation of a sorted expression encloses both the expression to be sorted and the sort fields in one <q:sortBy/> element. This is a significant restructuring of the query syntax, but it makes the scope of the sort clearer.

3.4 Example 4

Here is Q26 from the the [XQuery Working Draft] : "Using a recursive function, compute the maximum depth of the document named "partlist.xml."

```
NAMESPACE xsd = "http://www.w3.org/2001/XMLSchema"

FUNCTION depth (ELEMENT $e) RETURNS xsd:integer
{
    IF empty($e/*)
    THEN 1
    ELSE max(depth($e/*)) + 1
```

```
}

depth(document("partlist.xml"))
```

Here is the equivalent XML syntax.

```
<q:query xmlns:q="http://www.w3.org/Quilt"  xmlns:xsd="http://www.w3.org/2001/XMLSc

  <q:functionDefinition functionName="depth" datatype="xsd:integer">
    <q:argumentDeclaration name="$e" datatype="ELEMENT" />
    <q:ifThenElseExpr>
      <q:function name="empty">
        <q:step axis="CHILD">
          <q:variable>$e</q:variable>
          <q:identifier>*</q:identifier>
        </q:step>
      </q:function>
      <q:constant datatype="INTEGER">1</q:constant>
      <q:function name="PLUS">
        <q:function name="max">
          <q:function name="depth">
            <q:step axis="CHILD">
              <q:variable>$e</q:variable>
              <q:identifier>*</q:identifier>
            </q:step>
          </q:function>
        </q:function>
        <q:constant datatype="INTEGER">1</q:constant>
      </q:function>
    </q:ifThenElseExpr>
  </q:functionDefinition>

<!-- The function call: -->

  <q:function name="depth">
    <q:function name="document">
      <q:constant datatype="CHARSTRING">partlist.xml</q:constant>
    </q:function>
  </q:function>
</q:query>
```

In the above example, note the syntax used for function definitions and function calls.

4 An XML Schema for the XQuery XML Syntax

Here is the XML Schema for the proposed syntax.

```
<?xml version = "1.0" encoding = "UTF-8"?>
<xsd:schema xmlns:xsd = "http://www.w3.org/2001/XMLSchema">
    <xsd:group name = "expression">
        <xsd:choice>
            <xsd:element ref = "variable"/>
            <xsd:element ref = "constant"/>
            <xsd:element ref = "function"/>
            <xsd:element ref = "flwr"/>
            <xsd:element ref = "elementConstructor"/>
            <xsd:element ref = "predicatedExpr"/>
```

```
            <xsd:element ref = "sortBy"/>
            <xsd:element ref = "ifThenElseExpr"/>
            <xsd:element ref = "quantifier"/>
            <xsd:element ref = "exprList"/>
            <xsd:element ref = "step"/>
            <xsd:element ref = "identifier"/>
            <xsd:element ref = "nodeKindTest"/>
        </xsd:choice>
    </xsd:group>
    <xsd:element name = "query">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref = "functionDefinition" minOccurs = "0"
                        maxOccurs = "unbounded"/>
                <xsd:group ref = "expression"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name = "functionDefinition">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref = "argumentDeclaration" minOccurs = "0"
                    maxOccurs = "unbounded"/>
                <xsd:group ref = "expression"/>
            </xsd:sequence>
            <xsd:attribute name = "functionName" use = "required" type = "xsd:strir
            <xsd:attribute name = "datatype" use = "required" type = "xsd:string"/>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name = "argumentDeclaration">
        <xsd:complexType>
            <xsd:attribute name = "name" use = "required" type = "xsd:string"/>
            <xsd:attribute name = "datatype" use = "required" type = "xsd:string"/>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name = "exprList">
        <xsd:complexType>
            <xsd:choice minOccurs = "0" maxOccurs = "unbounded">
                <xsd:group ref = "expression"/>
            </xsd:choice>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name = "predicatedExpr">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:group ref = "expression"/>
                <xsd:element ref = "predicate" maxOccurs = "unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name = "predicate">
        <xsd:complexType>
            <xsd:choice>
                <xsd:sequence>
                    <xsd:element ref = "rangeFrom"/>
                    <xsd:element ref = "rangeTo"/>
                </xsd:sequence>
                <xsd:group ref = "expression"/>
            </xsd:choice>
        </xsd:complexType>
```

```
</xsd:element>
<xsd:element name = "rangeFrom">
    <xsd:complexType>
        <xsd:choice>
            <xsd:group ref = "expression"/>
        </xsd:choice>
    </xsd:complexType>
</xsd:element>
<xsd:element name = "rangeTo">
    <xsd:complexType>
        <xsd:choice>
            <xsd:group ref = "expression"/>
        </xsd:choice>
    </xsd:complexType>
</xsd:element>
<xsd:element name = "variable" type = "xsd:string"/>
<xsd:element name = "identifier" type = "xsd:string"/>
<xsd:element name = "constant">
    <xsd:complexType>
        <xsd:simpleContent>
            <xsd:extension base = "xsd:string">
                <xsd:attribute name = "datatype" type = "xsd:string"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
</xsd:element>
<xsd:element name = "function">
    <xsd:complexType>
        <xsd:choice minOccurs = "0" maxOccurs = "unbounded">
            <xsd:group ref = "expression"/>
        </xsd:choice>
        <xsd:attribute name = "name" use = "required" type = "xsd:string"/>
    </xsd:complexType>
</xsd:element>
<xsd:element name = "flwr">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:choice maxOccurs = "unbounded">
                <xsd:element ref = "forAssignment"/>
                <xsd:element ref = "letAssignment"/>
            </xsd:choice>
            <xsd:element ref = "where" minOccurs = "0"/>
            <xsd:element ref = "return"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name = "forAssignment">
    <xsd:complexType>
        <xsd:choice>
            <xsd:group ref = "expression"/>
        </xsd:choice>
        <xsd:attribute name = "variable" use = "required" type = "xsd:string"/>
    </xsd:complexType>
</xsd:element>
<xsd:element name = "letAssignment">
    <xsd:complexType>
        <xsd:choice>
            <xsd:group ref = "expression"/>
        </xsd:choice>
        <xsd:attribute name = "variable" use = "required" type = "xsd:string"/>
```

```
    </xsd:complexType>
</xsd:element>
<xsd:element name = "where">
    <xsd:complexType>
        <xsd:choice>
            <xsd:group ref = "expression"/>
        </xsd:choice>
    </xsd:complexType>
</xsd:element>
<xsd:element name = "return">
    <xsd:complexType>
        <xsd:choice>
            <xsd:group ref = "expression"/>
        </xsd:choice>
    </xsd:complexType>
</xsd:element>
<xsd:element name = "ifThenElseExpr">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:group ref = "expression"/>
            <xsd:group ref = "expression"/>
            <xsd:group ref = "expression"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name = "sortBy">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:group ref = "expression"/>
            <xsd:element ref = "sortfield" maxOccurs = "unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name = "sortfield">
    <xsd:complexType>
        <xsd:choice>
            <xsd:group ref = "expression"/>
        </xsd:choice>
        <xsd:attribute name = "order" use = "default" value = "ASCENDING">
            <xsd:simpleType>
                <xsd:restriction base = "xsd:NMTOKEN">
                    <xsd:enumeration value = "ASCENDING"/>
                    <xsd:enumeration value = "DESCENDING"/>
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:attribute>
    </xsd:complexType>
</xsd:element>
<xsd:element name = "quantifier">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref = "quantifierAssignment"/>
            <xsd:group ref = "expression"/>
        </xsd:sequence>
        <xsd:attribute name = "type" use = "default" value = "SOME">
            <xsd:simpleType>
                <xsd:restriction base = "xsd:NMTOKEN">
                    <xsd:enumeration value = "SOME"/>
                    <xsd:enumeration value = "EVERY"/>
                </xsd:restriction>
```

```
            </xsd:simpleType>
        </xsd:attribute>
    </xsd:complexType>
</xsd:element>
<xsd:element name = "quantifierAssignment">
    <xsd:complexType>
        <xsd:choice>
            <xsd:group ref = "expression"/>
        </xsd:choice>
        <xsd:attribute name = "variable" use = "required" type = "xsd:string"/>
    </xsd:complexType>
</xsd:element>
<xsd:element name = "elementConstructor">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref = "tagName"/>
            <xsd:element ref = "attributeConstructor" minOccurs = "0"
                maxOccurs = "unbounded"/>
            <xsd:choice minOccurs = "0" maxOccurs = "unbounded">
                <xsd:group ref = "expression"/>
            </xsd:choice>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name = "tagName">
    <xsd:complexType>
        <xsd:choice>
            <xsd:element ref = "identifier"/>
            <xsd:element ref = "variable"/>
        </xsd:choice>
    </xsd:complexType>
</xsd:element>
<xsd:element name = "attributeConstructor">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref = "attributeName"/>
            <xsd:element ref = "attributeValue"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name = "attributeName">
    <xsd:complexType>
        <xsd:choice>
            <xsd:element ref = "identifier"/>
            <xsd:element ref = "variable"/>
        </xsd:choice>
    </xsd:complexType>
</xsd:element>
<xsd:element name = "attributeValue">
    <xsd:complexType>
        <xsd:choice>
            <xsd:group ref = "expression"/>
        </xsd:choice>
    </xsd:complexType>
</xsd:element>
<xsd:element name = "step">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:group ref = "expression"/>
            <xsd:group ref = "expression"/>
```

```
                </xsd:sequence>
                <xsd:attribute name = "axis" use = "required">
                    <xsd:simpleType>
                        <xsd:restriction base = "xsd:NMTOKEN">
                            <xsd:enumeration value = "DEREFERENCE"/>
                            <xsd:enumeration value = "ANCESTOR"/>
                            <xsd:enumeration value = "ANCESTORORSELF"/>
                            <xsd:enumeration value = "ATTRIBUTE"/>
                            <xsd:enumeration value = "CHILD"/>
                            <xsd:enumeration value = "DESCENDANT"/>
                            <xsd:enumeration value = "DESCENDANTORSELF"/>
                            <xsd:enumeration value = "FOLLOWING"/>
                            <xsd:enumeration value = "FOLLOWINGSIBILING"/>
                            <xsd:enumeration value = "NAMESPACE"/>
                            <xsd:enumeration value = "PARENT"/>
                            <xsd:enumeration value = "PRECEDING"/>
                            <xsd:enumeration value = "PRECEDINGSIBLING"/>
                            <xsd:enumeration value = ""/>
                            <xsd:enumeration value = "SLASHSLASH"/>
                            <xsd:enumeration value = "SELF"/>
                        </xsd:restriction>
                    </xsd:simpleType>
                </xsd:attribute>
                <xsd:attribute name = "abbreviated" use = "default" value = "true">
                    <xsd:simpleType>
                        <xsd:restriction base = "xsd:NMTOKEN">
                            <xsd:enumeration value = "true"/>
                            <xsd:enumeration value = "false"/>
                        </xsd:restriction>
                    </xsd:simpleType>
                </xsd:attribute>
            </xsd:complexType>
        </xsd:element>
        <xsd:element name = "dot">
            <xsd:complexType/>
        </xsd:element>
        <xsd:element name = "dotdot">
            <xsd:complexType/>
        </xsd:element>
        <xsd:element name = "nodeKindTest">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element ref = "piTargetTest" minOccurs = "0"/>
                </xsd:sequence>
                <xsd:attribute name = "kind" use = "required">
                    <xsd:simpleType>
                        <xsd:restriction base = "xsd:NMTOKEN">
                            <xsd:enumeration value = "NODE"/>
                            <xsd:enumeration value = "TEXT"/>
                            <xsd:enumeration value = "COMMENT"/>
                            <xsd:enumeration value = "DATA"/>
                            <xsd:enumeration value = "PROCESSING_INSTRUCTION"/>
                        </xsd:restriction>
                    </xsd:simpleType>
                </xsd:attribute>
            </xsd:complexType>
        </xsd:element>
        <xsd:element name = "piTargetTest" type = "xsd:string"/>
    </xsd:schema>
```

5 A DTD for the XQuery XML Syntax

Here is the DTD for the proposed syntax. Again, productions from the[XQuery Working Draft] are shown as comments followed by equivalent DTD constructions.

Note that the examples shown above make heavy use of namespaces. Since DTDs do not support namespaces this limits the ability to validate XQueries using DTDs especially in the case of element construction.

```
<!ENTITY % ORDER_LITERALS "(ASCENDING | DESCENDING)">

<!ENTITY % QUANTIFIER_TYPE "(SOME | EVERY)">

<!ENTITY % expression "(q:variable | q:constant | q:function | q:flwr
 | q:elementConstructor | q:predicatedExpr | q:sortBy | q:ifThenElseExpr
 | q:quantifier | q:exprList | q:step | q:identifier | q:nodeKindTest)">

<!ENTITY % AXIS_TYPE "(DEREFERENCE | ANCESTOR | ANCESTORORSELF
| ATTRIBUTE | CHILD | DESCENDANT | DESCENDANTORSELF | FOLLOWING
| FOLLOWINGSIBILING | NAMESPACE | PARENT | PRECEDING
| PRECEDINGSIBLING | SLASHSLASH | SELF)">

<!ENTITY % NODE_KIND "(NODE | TEXT | COMMENT | DATA | PROCESSING_INSTRUCTION)">

<!ELEMENT q:query (q:functionDefinition* , %expression;)>

<!ELEMENT q:functionDefinition (q:argumentDeclaration* , %expression;)>
<!ATTLIST q:functionDefinition  functionName CDATA  #REQUIRED
                                datatype     CDATA  #REQUIRED >

<!ELEMENT q:argumentDeclaration EMPTY>
<!ATTLIST q:argumentDeclaration  name     CDATA  #REQUIRED
                                 datatype CDATA  #REQUIRED >

<!ELEMENT q:exprList (%expression;)*>

<!ELEMENT q:predicatedExpr (%expression; , q:predicate+)>

<!ELEMENT q:predicate ((q:rangeFrom , q:rangeTo) | %expression;)>

<!ELEMENT q:rangeFrom (%expression;)>

<!ELEMENT q:rangeTo (%expression;)>

<!ELEMENT q:variable (#PCDATA)>

<!ELEMENT q:identifier (#PCDATA)>

<!ELEMENT q:constant (#PCDATA)>
<!ATTLIST q:constant  datatype CDATA  #IMPLIED >

<!ELEMENT q:function (%expression;)*>
<!ATTLIST q:function  name CDATA  #REQUIRED >

<!ELEMENT q:flwr ((q:forAssignment | q:letAssignment)+ , q:where? , q:return)>

<!ELEMENT q:forAssignment %expression;>
```

```
<!ATTLIST q:forAssignment  variable CDATA  #REQUIRED >

<!ELEMENT q:letAssignment %expression;>
<!ATTLIST q:letAssignment  variable CDATA  #REQUIRED >

<!ELEMENT q:where (%expression;)>

<!ELEMENT q:return (%expression;)>

<!ELEMENT q:ifThenElseExpr (%expression; , %expression; , %expression;)>

<!ELEMENT q:sortBy (%expression; , q:sortfield+)>

<!ELEMENT q:sortfield (%expression;)>
<!ATTLIST q:sortfield  order %ORDER_LITERALS; "ASCENDING"     >

<!ELEMENT q:quantifier (q:quantifierAssignment , %expression;)>
<!ATTLIST q:quantifier  type %QUANTIFIER_TYPE; "SOME"     >

<!ELEMENT q:quantifierAssignment (%expression;)>
<!ATTLIST q:quantifierAssignment  variable CDATA  #REQUIRED >

<!ELEMENT q:elementConstructor (q:tagName , q:attributeConstructor* , (%expression;

<!ELEMENT q:tagName (q:identifier | q:variable)>

<!ELEMENT q:attributeConstructor (q:attributeName , q:attributeValue)>

<!ELEMENT q:attributeName (q:identifier | q:variable)>

<!ELEMENT q:attributeValue (%expression;)>

<!ELEMENT q:step (%expression; , %expression;)>
<!ATTLIST q:step  axis %AXIS_TYPE;  #REQUIRED
                  abbreviated  (true | false )  'true' >

<!ELEMENT q:dot EMPTY>

<!ELEMENT q:dotdot EMPTY>

<!ELEMENT q:nodeKindTest (q:piTargetTest?)>

<!ATTLIST q:nodeKindTest  kind %NODE_KIND; #REQUIRED     >
<!ELEMENT q:piTargetTest (#PCDATA)>
```

A References

XML Query 1.0 Requirements
World Wide Web Consortium. XML Query 1.0 Requirements. W3C Working Draft, 15 Feb 2001. See http://www.w3.org/TR/xmlquery-req.

XQuery Working Draft
XQuery 1.0: A Query Language for XML. See http://www.w3.org/TR/xquery/

XQuery 1.0 Formal Semantics
World Wide Web Consortium. XQuery 1.0 Formal Semantics. W3C Working Draft, 7 June 2001. See http://www.w3.org/TR/query-semantics/.

B XQuery Issues (Non-Normative)

This section contains the current issues for XQuery. The individual issues are shown in detail after an abbreviated issues list.

B.1 Issue List

Issue	Priority	Status	ID
1: Should path expressions be expanded? (xqueryx-expand-paths)			
2: Human readable XQueryX? (xqueryx-human-readable)			
3: Should XQueryX mirror the Formal Semantics? (xqueryx-semantics)			
4: XQueryX and XSLT (xqueryx-xslt)			
5: XQueryX Operator Syntax (xqueryx-operator-syntax)			
6: Define Transformations from XQuery Grammar (xqueryx-define-transformation)			
7: Representing Path Expressions (xqueryx-path-expressions)			
8: Representing // (xqueryx-slash-slash)			
9: Symbol space for operators (xqueryx-operators-symbolspace)			
10: Wildcard (xqueryx-star-identifier)			
11: Type Expressions (xqueryx-cast-treat-typeswitch)			
12: SCHEMA declaration (xqueryx-schema-declarations)			
13: Case of Constants (xqueryx-case-of-constants)			

B.2 XQueryX Issues

Issue 1 : Should path expressions be expanded? (xqueryx-expand-paths)

Originator: XMLQuery
Locus: xqueryx

Description:

The current representation requires path expressions to be expanded. Should this be optional?

Issue 2 : Human readable XQueryX? (xqueryx-human-readable)

Originator: XMLQuery
Locus: xqueryx

Description:

Should making XQueryX queries easier for humans to read and write be a goal?

Issue 3 : Should XQueryX mirror the Formal Semantics? (xqueryx-semantics)

Originator: XMLQuery
Locus: xqueryx

Description:

Should XQueryX mirror rather than the abstract syntax of XQuery?

Issue 4 : XQueryX and XSLT (xqueryx-xslt)

Originator: XMLQuery
Locus: xqueryx

Description:

Should there be some clear relationship between XQueryX and XSLT?

Issue 5 : XQueryX Operator Syntax (xqueryx-operator-syntax)

Originator: XMLQuery
Locus: xqueryx

Description:

XQueryX currently uses function syntax for operators:

```
<q:function name="PLUS">
    <q:constant datatype="INTEGER">1</q:constant>
    <q:constant datatype="INTEGER">2</q:constant>
</q:function>
```

Several other possibilities have been suggested, eg:

```
<q:and/> and <q:equal/>
```

or

```
<q:binaryOperator name="AND"/>
<q:binaryOperator name="EQUALS"/>
```

or

```
<q:mathOperator name="and"/> and <q:logicalOperator name="equal"/>
```

Issue 6 : Define Transformations from XQuery Grammar (xqueryx-define-transformation)

Originator: XMLQuery
Locus: xqueryx

Description:

A well defined mapping from the XQuery grammar productions to XQueryX should be defined. This is more cost effective when both XQuery and XQueryX have settled down a bit more.

Issue 7 : Representing Path Expressions (xqueryx-path-expressions)

Originator: XMLQuery
Locus: xqueryx

Description:

Should path expressions be represented in a linear fashion, with the individual steps represented as a sequence? One Working Group member has suggested that the path "doc("a")/b/c" should be represented as follows:

```
<path>
    <function name="doc">
        <constant>a</constant>
    </function>
    <step axis="descendant">
        <identifier>b</identifier>
    </step>
    <step axis="descendant">
        <identifier>c</identifier>
    </step>
</path>
```

Issue 8 : Representing // (xqueryx-slash-slash)

Originator: XMLQuery
Locus: xqueryx

Description:

In this document we use the constant SLASHSLASH as a fictive axis that corresponds to the meaning of the "//" operator. In fact, this operator does not correspond directly to an axis, but to descendant-or-self::node()/. For example, //para is short for /descendant-or-self::node()/child::para. Since XQueryX must be able to translate back to the original XQuery syntax, we currently use SLASHSLASH to preserve the original operator.

Is this the best way to represent the // operator in XQueryX?

Issue 9 : Symbol space for operators (xqueryx-operators-symbolspace)

Originator: XMLQuery
Locus: xqueryx

Description:

The current syntax does not distinguish functions from operators. Since a function may have the same name as an operator, some mechanism must be chosen to disambiguate their names. For instance, since a user can write a function named plus, perhaps the operator plus should be represented as:

```
<operator name="plus"/>
```

rather than

```
<function name="plus"/>
```

Issue 10 : Wildcard (xqueryx-star-identifier)

Originator: XMLQuery
Locus: xqueryx

Description:

In Example 4, should * be used as an identifier? It should probably be a wildcard.

Issue 11 : Type Expressions (xqueryx-cast-treat-typeswitch)

Originator: XMLQuery
Locus: xqueryx

Description:

CAST, TREAT, and TYPESWITCH have no representation in this Working Draft. This must be added soon.

Issue 12 : SCHEMA declaration (xqueryx-schema-declarations)

Originator: XMLQuery
Locus: xqueryx

Description:

There is currently no representation for schema declarations in XQueryX. This must be added.

Issue 13 : Case of Constants (xqueryx-case-of-constants)

Originator: XMLQuery
Locus: xqueryx

Description:

Should XQueryX constants use upper or lower case?