XML Syntax for XQuery 1.0 (XQueryX)

W3C Recommendation 23 January 2007

This version:

http://www.w3.org/TR/2007/REC-xqueryx-20070123

Latest version:

http://www.w3.org/TR/xqueryx

Previous version:

http://www.w3.org/TR/2006/PR-xqueryx-20061121

Authors and Contributors:

Jim Melton (Oracle) <jim.melton@oracle.com>
Subramanian Muralidhar (Microsoft)

Abstract

This document defines an XML Syntax for [XQuery 1.0: An XML Query Language].

Status of this document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.

This is one document in a set of eight documents that have progressed to Recommendation together (XQuery 1.0, XQueryX 1.0, XSLT 2.0, Data Model, Functions and Operators, Formal Semantics, Serialization, XPath 2.0).

This is a Recommendation of the W3C. It has been developed by the W3C XML Query Working Group, which is part of the XML Activity.

This document has been reviewed by W3C Members, by software developers, and by other W3C groups and interested parties, and is endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

This document incorporates minor changes made against the Proposed Recommendation of 21 November 2006; please see the public disposition of comments for details. Changes to this document since the Proposed Recommendation are detailed in the Appendix D – Change log on page 87.

Please report errors in this document using W3C's public Bugzilla system (instructions can be found at http://www.w3.org/XML/2005/04/qt-bugzilla). If access to that system is not feasible, you may send your comments to the W3C XSLT/XPath/XQuery public comments mailing list, public-qt-comments@w3.org. It will be very helpful if you include the string "[XQX]" in the subject line of your report, whether made in Bugzilla or in email. Each Bugzilla entry and email message should contain only one error report. Archives of the comments and responses are available at http://lists.w3.org/Archives/Public/public-qt-comments/.

This document was produced by a group operating under the 5 February 2004 W3C Patent Policy. W3C maintains a public list of any patent disclosures made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) must disclose the information in accordance with section 6 of the W3C Patent Policy.

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 grammar 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.

Because the two syntaxes are merely different grammars that express the same query semantics, they share all aspects of an XQuery processing system except for the component that recognizes and translates the source representation of a query (that is, the parser). The aspects that are shared include both the static context and the dynamic context that are defined in [XQuery 1.0: An XML Query Language].

There are several environments in which XQueryX may be useful:

- *Parser Reuse.* In heterogeneous 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 FLWOR 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 ordinary XML tools can be used.
- *Embedding Queries in XML.* XQueryX can be embedded directly in an XML document.

The most recent versions of the XQueryX XML Schema and the XQueryX XSLT stylesheet are available at http://www.w3.org/2005/XQueryX/xqueryx.xsd and http://www.w3.org/2005/XQueryX/xqueryx.xsl, respectively.

2. Mapping the XQuery Syntax

XQueryX is an XML representation of the abstract syntax found in Appendix A of [XQuery 1.0: An XML Query Language]. The XQueryX syntax is defined by the XQueryX Schema, which is specified in § 4 – An XML Schema for the XQuery XML Syntax on page 26. The XQueryX semantics are defined by a stylesheet that maps an instance of XQueryX to an instance of XQuery; see Appendix B – Transforming XQueryX to XQuery on page 56.

The main data structure in the XQueryX Schema is the set of types that describe expressions. We have chosen to model expressions using substitution groups, with an "expr" abstract base class and a number of intermediate abstract classes.

Consider the following productions from the abstract syntax:

```
FLWORExpr    ::=    (ForClause  | LetClause)+ WhereClause? OrderByClause? "return"
ExprSingle
ForClause    ::=    "for" "$" VarName TypeDeclaration? PositionalVar? "in"
```

```
                    ExprSingle ("," "$" VarName TypeDeclaration? PositionalVar?
                    "in" ExprSingle)*
LetClause    ::=    "let" "$" VarName TypeDeclaration? ":=" ExprSingle
                    ("," "$" VarName TypeDeclaration? ":=" ExprSingle)*
WhereClause  ::=    "where" ExprSingle
```

The following XQueryX Schema definitions reflect the structure of those productions from that abstract syntax:

```
<!-- The base class -->
  <xsd:complexType name="expr"/>
  <xsd:element name="expr" type="expr" abstract="true"/>

  <!-- Simple wrapper class -->
  <xsd:complexType name="exprWrapper">
    <xsd:sequence>
      <xsd:element ref="expr"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="flworExpr">
    <xsd:complexContent>
      <xsd:extension base="expr">
        <xsd:sequence>
          <xsd:choice maxOccurs="unbounded">
            <xsd:element ref="forClause"/>
            <xsd:element ref="letClause"/>
          </xsd:choice>
          <xsd:element name="whereClause" minOccurs="0"/>
          <xsd:element name="orderByClause" minOccurs="0"/>
          <xsd:element name="returnClause"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="flworExpr" type="flworExpr" substitutionGroup="expr"/>

  <xsd:element name="forClause">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="forClauseItem" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
```

```
<xsd:element name="forClauseItem">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="typedVariableBinding"/>
      <xsd:element ref="positionalVariableBinding" minOccurs="0"/>
      <xsd:element name="forExpr" type="exprWrapper"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="letClause">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="letClauseItem" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="letClauseItem">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="typedVariableBinding"/>
      <xsd:element name="letExpr" type="exprWrapper"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="whereClause" type="exprWrapper"/>

<xsd:element name="returnClause" type="exprWrapper"/>
```

Since XQuery uses the expression production liberally to allow expressions to be flexibly combined, XQueryX uses the exprWrapper type in embedded contexts to allow all expression types to occur.

3. Examples from the XML Query Use Cases in XML Syntax

Three of following examples are based on the data and queries in the XMP (Experiences and Exemplars) use case in [XML Query Use Cases], while a fourth example is based on the data and queries in the NS (Queries Using Namespaces) use case. For each example, we show the English description of the query, the XQuery solution given in [XML Query Use Cases], an XQueryX solution, and the XQuery that results from applying the XQueryX-to-XQuery transformation defined by the stylesheet in Appendix B – Transforming XQueryX to XQuery on page 56 to the XQueryX solution. That produced XQuery is presented only as a sanity-check — the intent of the stylesheet is not to create the identical XQuery expression given in [XML Query Use Cases], but to produce *a* valid XQuery expression with the same semantics. The semantics of the XQueryX solution are determined by the semantics of the XQuery expression that results

from that transformation. The "correctness" of that transformation is determined by asking the following the question: Can some XQueryX processor QX process some XQueryX document D1 to produce results R1, after which the stylesheet is used to translate D1 into an XQuery expression E1 that, when processed by some XQuery processor Q, produces results R2 that are equivalent (under some meaningful definition of "equivalent") to results R1?

Comparison of the results of the XQueryX-to-XQuery transformation given in this document with the XQuery solutions in the [XML Query Use Cases] may be helpful in evaluating the correctness of the XQueryX solution in each example.

The XQuery Use Cases solution given for each example is provided only to assist readers of this document in understanding the XQueryX solution. There is no intent to imply that this document specifies a "compilation" or "transformation" of XQuery syntax into XQueryX syntax.

In the following examples, 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 and manipulation. XQueryX is designed as a language that is convenient for production and modification by software, and not as a convenient syntax for humans to read and write.

Finally, please note that white space, including new lines, have been added to some of the XQueryX documents and XQuery expressions for readability. That additional white space is not produced by the XQueryX-to-XQuery transformation.

3.1. Example 1

Here is Q1 from the [XML Query Use Cases], use case XMP (Experiences and Exemplars): "List books published by Addison-Wesley after 1991, including their year and title."

3.1.1. XQuery solution in XQuery Use Cases:

```
<bib>
 {
  for $b in doc("http://bstore1.example.com/bib.xml")/bib/book
  where $b/publisher = "Addison-Wesley" and $b/@year > 1991
  return
    <book year="{ $b/@year }">
     { $b/title }
    </book>
 }
</bib>
```

3.1.2. A Solution in XQueryX:

```
<?xml version="1.0"?>
<xqx:module xmlns:xqx="http://www.w3.org/2005/XQueryX"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://www.w3.org/2005/XQueryX
                                http://www.w3.org/2005/XQueryX/xqueryx.xsd">
  <xqx:mainModule>
```

```
    <xqx:queryBody>
      <xqx:elementConstructor>
        <xqx:tagName>bib</xqx:tagName>
        <xqx:elementContent>
          <xqx:flworExpr>
            <xqx:forClause>
              <xqx:forClauseItem>
                <xqx:typedVariableBinding>
                  <xqx:varName>b</xqx:varName>
                </xqx:typedVariableBinding>
                <xqx:forExpr>
                  <xqx:pathExpr>
                    <xqx:stepExpr>
                      <xqx:filterExpr>
                        <xqx:functionCallExpr>
                          <xqx:functionName>doc</xqx:functionName>
                          <xqx:arguments>
                            <xqx:stringConstantExpr>

<xqx:value>http://bstore1.example.com/bib.xml</xqx:value>
                            </xqx:stringConstantExpr>
                          </xqx:arguments>
                        </xqx:functionCallExpr>
                      </xqx:filterExpr>
                    </xqx:stepExpr>
                    <xqx:stepExpr>
                      <xqx:xpathAxis>child</xqx:xpathAxis>
                      <xqx:nameTest>bib</xqx:nameTest>
                    </xqx:stepExpr>
                    <xqx:stepExpr>
                      <xqx:xpathAxis>child</xqx:xpathAxis>
                      <xqx:nameTest>book</xqx:nameTest>
                    </xqx:stepExpr>
                  </xqx:pathExpr>
                </xqx:forExpr>
              </xqx:forClauseItem>
            </xqx:forClause>
            <xqx:whereClause>
              <xqx:andOp>
                <xqx:firstOperand>
                  <xqx:equalOp>
                    <xqx:firstOperand>
                      <xqx:pathExpr>
                        <xqx:stepExpr>
                          <xqx:filterExpr>
```

```
                    <xqx:varRef>
                      <xqx:name>b</xqx:name>
                    </xqx:varRef>
                  </xqx:filterExpr>
                </xqx:stepExpr>
                <xqx:stepExpr>
                  <xqx:xpathAxis>child</xqx:xpathAxis>
                  <xqx:nameTest>publisher</xqx:nameTest>
                </xqx:stepExpr>
              </xqx:pathExpr>
            </xqx:firstOperand>
            <xqx:secondOperand>
              <xqx:stringConstantExpr>
                <xqx:value>Addison-Wesley</xqx:value>
              </xqx:stringConstantExpr>
            </xqx:secondOperand>
          </xqx:equalOp>
        </xqx:firstOperand>
        <xqx:secondOperand>
          <xqx:greaterThanOp>
            <xqx:firstOperand>
              <xqx:pathExpr>
                <xqx:stepExpr>
                  <xqx:filterExpr>
                    <xqx:varRef>
                      <xqx:name>b</xqx:name>
                    </xqx:varRef>
                  </xqx:filterExpr>
                </xqx:stepExpr>
                <xqx:stepExpr>
                  <xqx:xpathAxis>attribute</xqx:xpathAxis>
                  <xqx:nameTest>year</xqx:nameTest>
                </xqx:stepExpr>
              </xqx:pathExpr>
            </xqx:firstOperand>
            <xqx:secondOperand>
              <xqx:integerConstantExpr>
                <xqx:value>1991</xqx:value>
              </xqx:integerConstantExpr>
            </xqx:secondOperand>
          </xqx:greaterThanOp>
        </xqx:secondOperand>
      </xqx:andOp>
    </xqx:whereClause>
    <xqx:returnClause>
```

```
            <xqx:elementConstructor>
              <xqx:tagName>book</xqx:tagName>
              <xqx:attributeList>
                <xqx:attributeConstructor>
                  <xqx:attributeName>year</xqx:attributeName>
                  <xqx:attributeValueExpr>
                    <xqx:pathExpr>
                      <xqx:stepExpr>
                        <xqx:filterExpr>
                          <xqx:varRef>
                            <xqx:name>b</xqx:name>
                          </xqx:varRef>
                        </xqx:filterExpr>
                      </xqx:stepExpr>
                      <xqx:stepExpr>
                        <xqx:xpathAxis>attribute</xqx:xpathAxis>
                        <xqx:nameTest>year</xqx:nameTest>
                      </xqx:stepExpr>
                    </xqx:pathExpr>
                  </xqx:attributeValueExpr>
                </xqx:attributeConstructor>
              </xqx:attributeList>
              <xqx:elementContent>
                <xqx:pathExpr>
                  <xqx:stepExpr>
                    <xqx:filterExpr>
                      <xqx:varRef>
                        <xqx:name>b</xqx:name>
                      </xqx:varRef>
                    </xqx:filterExpr>
                  </xqx:stepExpr>
                  <xqx:stepExpr>
                    <xqx:xpathAxis>child</xqx:xpathAxis>
                    <xqx:nameTest>title</xqx:nameTest>
                  </xqx:stepExpr>
                </xqx:pathExpr>
              </xqx:elementContent>
            </xqx:elementConstructor>
          </xqx:returnClause>
        </xqx:flworExpr>
      </xqx:elementContent>
    </xqx:elementConstructor>
  </xqx:queryBody>
 </xqx:mainModule>
</xqx:module>
```

3.1.3. Transformation of XQueryX Solution into XQuery

Application of the stylesheet in Appendix B – Transforming XQueryX to XQuery on page 56 to the XQueryX solution results in:

```
<bib>{
 for $b in doc("http://bstore1.example.com/bib.xml")/child::bib/child::book
 where (($b/child::publisher = "Addison-Wesley") and ($b/attribute::year > 1991))
 return <book year="{$b/attribute::year}">{$b/child::title}</book>
}</bib>
```

3.1.4. Corresponding Grammar Abstract Parse Tree

For comparison, here is the abstract parse tree corresponding to the XQuery for Example 1, as produced by the XQuery grammar applets found at http://www.w3.org/2005/qt-applets/xqueryApplet.html.

```
XPath2
  QueryList
    Module
      MainModule
        Prolog
        QueryBody
          Expr
            PathExpr
              Constructor
                DirectConstructor
                  DirElemConstructor >
                    LessThanOpOrTagO <
                    TagQName bib
                    DirAttributeList
                    StartTagClose >
                    DirElemContent
                      ElementContentChar
                    DirElemContent
                      ElementContentChar
                    DirElemContent
                      CommonContent
                        EnclosedExpr
                          Lbrace {
                          Expr
                            FLWORExpr
                              ForClause
                                VarName
                                  QName b
                                PathExpr
                                  FunctionCall
                                    FunctionQName doc
```

```
                                    PathExpr
                                      StringLiteral
"http://bstore1.example.com/bib.xml"
                                  StepExpr
                                    AbbrevForwardStep
                                      NodeTest
                                        NameTest
                                          QName bib
                                  StepExpr
                                    AbbrevForwardStep
                                      NodeTest
                                        NameTest
                                          QName book
                                WhereClause
                                  AndExpr and
                                    ComparisonExpr =
                                      PathExpr
                                        VarName
                                          QName b
                                        StepExpr
                                          AbbrevForwardStep
                                            NodeTest
                                              NameTest
                                                QName publisher
                                      PathExpr
                                        StringLiteral "Addison-Wesley"
                                    ComparisonExpr >
                                      PathExpr
                                        VarName
                                          QName b
                                        StepExpr
                                          AbbrevForwardStep @
                                            NodeTest
                                              NameTest
                                                QName year
                                      PathExpr
                                        IntegerLiteral 1991
                                PathExpr
                                  Constructor
                                    DirectConstructor
                                      DirElemConstructor >
                                        LessThanOpOrTagO <
                                        TagQName book
                                        DirAttributeList
                                          S
```

```
  TagQName year
  ValueIndicator =
  DirAttributeValue
    OpenQuot "
    QuotAttrValueContent
      CommonContent
        EnclosedExpr
          Lbrace {
          Expr
            PathExpr
              VarName
                QName b
              StepExpr
                AbbrevForwardStep @
                  NodeTest
                    NameTest
                      QName year
          Rbrace }
    CloseQuot "
StartTagClose >
DirElemContent
  ElementContentChar
DirElemContent
  ElementContentChar
DirElemContent
  ElementContentChar
DirElemContent
  ElementContentChar
DirElemContent
  ElementContentChar
DirElemContent
  ElementContentChar
DirElemContent
  CommonContent
    EnclosedExpr
      Lbrace {
      Expr
        PathExpr
          VarName
            QName b
          StepExpr
            AbbrevForwardStep
              NodeTest
                NameTest
                  QName title
```

```
                                        Rbrace }
                                    DirElemContent
                                      ElementContentChar
                                    DirElemContent
                                      ElementContentChar
                                    DirElemContent
                                      ElementContentChar
                                    DirElemContent
                                      ElementContentChar
                                    DirElemContent </
                                      ElementContentChar
                          Rbrace }
                      DirElemContent </
                        ElementContentChar
```

3.2. Example 2

Here is Q4 from the [XML Query Use Cases], use case XMP (Experiences and Exemplars): "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."

3.2.1. XQuery solution in XQuery Use Cases:

```
<results>
  {
    let $a := doc("http://bstore1.example.com/bib/bib.xml")//author
    for $last in distinct-values($a/last),
        $first in distinct-values($a[last=$last]/first)
    order by $last, $first
    return
        <result>
            <author>
               <last>{ $last }</last>
               <first>{ $first }</first>
            </author>
            {
                for $b in doc("http://bstore1.example.com/bib.xml")/bib/book
                where some $ba in $b/author
                      satisfies ($ba/last = $last and $ba/first=$first)
                return $b/title
            }
        </result>
  }
</results>
```

3.2.2. A solution in XQueryX:

```
<?xml version="1.0"?>
<xqx:module xmlns:xqx="http://www.w3.org/2005/XQueryX"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://www.w3.org/2005/XQueryX
                        http://www.w3.org/2005/XQueryX/xqueryx.xsd">
  <xqx:mainModule>
    <xqx:queryBody>
      <xqx:elementConstructor>
        <xqx:tagName>results</xqx:tagName>
        <xqx:elementContent>
          <xqx:flworExpr>
            <xqx:letClause>
              <xqx:letClauseItem>
                <xqx:typedVariableBinding>
                  <xqx:varName>a</xqx:varName>
                </xqx:typedVariableBinding>
                <xqx:letExpr>
                 <xqx:pathExpr>
                   <xqx:stepExpr>
                     <xqx:filterExpr>
                       <xqx:functionCallExpr>
                         <xqx:functionName>doc</xqx:functionName>
                         <xqx:arguments>
                           <xqx:stringConstantExpr>
                          <xqx:value>http://bstore1.example.com/bib.xml</xqx:value>

                           </xqx:stringConstantExpr>
                         </xqx:arguments>
                       </xqx:functionCallExpr>
                     </xqx:filterExpr>
                   </xqx:stepExpr>
                   <xqx:stepExpr>
                     <xqx:xpathAxis>descendant-or-self</xqx:xpathAxis>
                     <xqx:anyKindTest/>
                   </xqx:stepExpr>
                   <xqx:stepExpr>
                     <xqx:xpathAxis>child</xqx:xpathAxis>
                     <xqx:nameTest>author</xqx:nameTest>
                   </xqx:stepExpr>
                 </xqx:pathExpr>
                </xqx:letExpr>
              </xqx:letClauseItem>
            </xqx:letClause>
```

```
<xqx:forClause>
  <xqx:forClauseItem>
    <xqx:typedVariableBinding>
      <xqx:varName>last</xqx:varName>
    </xqx:typedVariableBinding>
    <xqx:forExpr>
      <xqx:functionCallExpr>
        <xqx:functionName>distinct-values</xqx:functionName>
        <xqx:arguments>
          <xqx:pathExpr>
            <xqx:stepExpr>
              <xqx:filterExpr>
                <xqx:varRef>
                  <xqx:name>a</xqx:name>
                </xqx:varRef>
              </xqx:filterExpr>
            </xqx:stepExpr>
            <xqx:stepExpr>
              <xqx:xpathAxis>child</xqx:xpathAxis>
              <xqx:nameTest>last</xqx:nameTest>
            </xqx:stepExpr>
          </xqx:pathExpr>
        </xqx:arguments>
      </xqx:functionCallExpr>
    </xqx:forExpr>
  </xqx:forClauseItem>
  <xqx:forClauseItem>
    <xqx:typedVariableBinding>
      <xqx:varName>first</xqx:varName>
    </xqx:typedVariableBinding>
    <xqx:forExpr>
      <xqx:functionCallExpr>
        <xqx:functionName>distinct-values</xqx:functionName>
        <xqx:arguments>
          <xqx:pathExpr>
            <xqx:stepExpr>
              <xqx:filterExpr>
                <xqx:varRef>
                  <xqx:name>a</xqx:name>
                </xqx:varRef>
              </xqx:filterExpr>
              <xqx:predicates>
                <xqx:equalOp>
                  <xqx:firstOperand>
                    <xqx:pathExpr>
```

```
                        <xqx:stepExpr>
                          <xqx:filterExpr>
                            <xqx:contextItemExpr/>
                          </xqx:filterExpr>
                        </xqx:stepExpr>
                        <xqx:stepExpr>
                          <xqx:xpathAxis>child</xqx:xpathAxis>
                          <xqx:nameTest>last</xqx:nameTest>
                        </xqx:stepExpr>
                      </xqx:pathExpr>
                    </xqx:firstOperand>
                    <xqx:secondOperand>
                      <xqx:varRef>
                        <xqx:name>last</xqx:name>
                      </xqx:varRef>
                    </xqx:secondOperand>
                  </xqx:equalOp>
                </xqx:predicates>
              </xqx:stepExpr>
              <xqx:stepExpr>
                <xqx:xpathAxis>child</xqx:xpathAxis>
                <xqx:nameTest>first</xqx:nameTest>
              </xqx:stepExpr>
            </xqx:pathExpr>
          </xqx:arguments>
        </xqx:functionCallExpr>
      </xqx:forExpr>
    </xqx:forClauseItem>
  </xqx:forClause>
  <xqx:orderByClause>
    <xqx:orderBySpec>
      <xqx:orderByExpr>
        <xqx:varRef>
          <xqx:name>last</xqx:name>
        </xqx:varRef>
      </xqx:orderByExpr>
    </xqx:orderBySpec>
    <xqx:orderBySpec>
      <xqx:orderByExpr>
        <xqx:varRef>
          <xqx:name>first</xqx:name>
        </xqx:varRef>
      </xqx:orderByExpr>
    </xqx:orderBySpec>
  </xqx:orderByClause>
```

```
            <xqx:returnClause>
              <xqx:elementConstructor>
                <xqx:tagName>result</xqx:tagName>
                <xqx:elementContent>
                  <xqx:elementConstructor>
                    <xqx:tagName>author</xqx:tagName>
                    <xqx:elementContent>
                      <xqx:elementConstructor>
                        <xqx:tagName>last</xqx:tagName>
                        <xqx:elementContent>
                          <xqx:varRef>
                            <xqx:name>last</xqx:name>
                          </xqx:varRef>
                        </xqx:elementContent>
                      </xqx:elementConstructor>
                      <xqx:elementConstructor>
                        <xqx:tagName>first</xqx:tagName>
                        <xqx:elementContent>
                          <xqx:varRef>
                            <xqx:name>first</xqx:name>
                          </xqx:varRef>
                        </xqx:elementContent>
                      </xqx:elementConstructor>
                    </xqx:elementContent>
                  </xqx:elementConstructor>
                    <xqx:flworExpr>
                      <xqx:forClause>
                        <xqx:forClauseItem>
                          <xqx:typedVariableBinding>
                            <xqx:varName>b</xqx:varName>
                          </xqx:typedVariableBinding>
                          <xqx:forExpr>
                            <xqx:pathExpr>
                              <xqx:stepExpr>
                                <xqx:filterExpr>
                                  <xqx:functionCallExpr>
                                    <xqx:functionName>doc</xqx:functionName>
                                    <xqx:arguments>
                                      <xqx:stringConstantExpr>

<xqx:value>http://bstore1.example.com/bib.xml</xqx:value>
                                      </xqx:stringConstantExpr>
                                    </xqx:arguments>
                                  </xqx:functionCallExpr>
                                </xqx:filterExpr>
```

```
            </xqx:stepExpr>
            <xqx:stepExpr>
              <xqx:xpathAxis>child</xqx:xpathAxis>
              <xqx:nameTest>bib</xqx:nameTest>
            </xqx:stepExpr>
            <xqx:stepExpr>
              <xqx:xpathAxis>child</xqx:xpathAxis>
              <xqx:nameTest>book</xqx:nameTest>
            </xqx:stepExpr>
          </xqx:pathExpr>
        </xqx:forExpr>
      </xqx:forClauseItem>
    </xqx:forClause>
    <xqx:whereClause>
      <xqx:quantifiedExpr>
        <xqx:quantifier>some</xqx:quantifier>
        <xqx:quantifiedExprInClause>
          <xqx:typedVariableBinding>
            <xqx:varName>ba</xqx:varName>
          </xqx:typedVariableBinding>
          <xqx:sourceExpr>
            <xqx:pathExpr>
              <xqx:stepExpr>
                <xqx:filterExpr>
                  <xqx:varRef>
                    <xqx:name>b</xqx:name>
                  </xqx:varRef>
                </xqx:filterExpr>
              </xqx:stepExpr>
              <xqx:stepExpr>
                <xqx:xpathAxis>child</xqx:xpathAxis>
                <xqx:nameTest>author</xqx:nameTest>
              </xqx:stepExpr>
            </xqx:pathExpr>
          </xqx:sourceExpr>
        </xqx:quantifiedExprInClause>
        <xqx:predicateExpr>
          <xqx:andOp>
            <xqx:firstOperand>
              <xqx:equalOp>
                <xqx:firstOperand>
                  <xqx:pathExpr>
                    <xqx:stepExpr>
                      <xqx:filterExpr>
                        <xqx:varRef>
```

```
                                    <xqx:name>ba</xqx:name>
                                  </xqx:varRef>
                                </xqx:filterExpr>
                              </xqx:stepExpr>
                              <xqx:stepExpr>
                                <xqx:xpathAxis>child</xqx:xpathAxis>
                                <xqx:nameTest>last</xqx:nameTest>
                              </xqx:stepExpr>
                            </xqx:pathExpr>
                          </xqx:firstOperand>
                          <xqx:secondOperand>
                            <xqx:varRef>
                              <xqx:name>last</xqx:name>
                            </xqx:varRef>
                          </xqx:secondOperand>
                        </xqx:equalOp>
                      </xqx:firstOperand>
                      <xqx:secondOperand>
                        <xqx:equalOp>
                          <xqx:firstOperand>
                            <xqx:pathExpr>
                              <xqx:stepExpr>
                                <xqx:filterExpr>
                                  <xqx:varRef>
                                    <xqx:name>ba</xqx:name>
                                  </xqx:varRef>
                                </xqx:filterExpr>
                              </xqx:stepExpr>
                              <xqx:stepExpr>
                                <xqx:xpathAxis>child</xqx:xpathAxis>
                                <xqx:nameTest>first</xqx:nameTest>
                              </xqx:stepExpr>
                            </xqx:pathExpr>
                          </xqx:firstOperand>
                          <xqx:secondOperand>
                            <xqx:varRef>
                              <xqx:name>first</xqx:name>
                            </xqx:varRef>
                          </xqx:secondOperand>
                        </xqx:equalOp>
                      </xqx:secondOperand>
                    </xqx:andOp>
                  </xqx:predicateExpr>
                </xqx:quantifiedExpr>
              </xqx:whereClause>
```

```
                    <xqx:returnClause>
                      <xqx:pathExpr>
                        <xqx:stepExpr>
                          <xqx:filterExpr>
                            <xqx:varRef>
                              <xqx:name>b</xqx:name>
                            </xqx:varRef>
                          </xqx:filterExpr>
                        </xqx:stepExpr>
                        <xqx:stepExpr>
                          <xqx:xpathAxis>child</xqx:xpathAxis>
                          <xqx:nameTest>title</xqx:nameTest>
                        </xqx:stepExpr>
                      </xqx:pathExpr>
                    </xqx:returnClause>
                  </xqx:flworExpr>
                </xqx:elementContent>
              </xqx:elementConstructor>
            </xqx:returnClause>
          </xqx:flworExpr>
        </xqx:elementContent>
      </xqx:elementConstructor>
    </xqx:queryBody>
  </xqx:mainModule>
</xqx:module>
```

3.2.3. Transformation of XQueryX Solution into XQuery

Application of the stylesheet in Appendix B – Transforming XQueryX to XQuery on page 56 to the XQueryX solution results in:

```
<results>{
 let
$a:=doc("http://bstore1.example.com/bib.xml")/descendant-or-self::node()/child::author

 for $last in distinct-values($a/child::last), $first in
distinct-values($a[(./child::last = $last)]/child::first)
 order by $last , $first
 return <result><author><last>{$last}</last><first>{$first}</first></author>{
 for $b in doc("http://bstore1.example.com/bib.xml")/child::bib/child::book
 where (some $ba in $b/child::author satisfies (($ba/child::last = $last) and
($ba/child::first = $first)))
 return $b/child::title
}</result>
}</results>
```

3.3. Example 3

Here is Q7 from the [XML Query Use Cases], use case XMP (Experiences and Exemplars): "List the titles and years of all books published by Addison-Wesley after 1991, in alphabetic order."

3.3.1. XQuery solution in XQuery Use Cases:

```
<bib>
  {
    for $b in doc("http://bstore1.example.com/bib.xml")//book
    where $b/publisher = "Addison-Wesley" and $b/@year > 1991
    order by $b/title
    return
        <book>
            { $b/@year }
            { $b/title }
        </book>
  }
</bib>
```

3.3.2. A solution in XQueryX:

```
<?xml version="1.0"?>
<xqx:module xmlns:xqx="http://www.w3.org/2005/XQueryX"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://www.w3.org/2005/XQueryX
                                http://www.w3.org/2005/XQueryX/xqueryx.xsd">
  <xqx:mainModule>
    <xqx:queryBody>
      <xqx:elementConstructor>
        <xqx:tagName>bib</xqx:tagName>
        <xqx:elementContent>
          <xqx:flworExpr>
            <xqx:forClause>
              <xqx:forClauseItem>
                <xqx:typedVariableBinding>
                  <xqx:varName>b</xqx:varName>
                </xqx:typedVariableBinding>
                <xqx:forExpr>
                  <xqx:pathExpr>
                    <xqx:stepExpr>
                      <xqx:filterExpr>
                        <xqx:functionCallExpr>
                          <xqx:functionName>doc</xqx:functionName>
                          <xqx:arguments>
                            <xqx:stringConstantExpr>
```

```
<xqx:value>http://bstore1.example.com/bib.xml</xqx:value>
                          </xqx:stringConstantExpr>
                        </xqx:arguments>
                      </xqx:functionCallExpr>
                    </xqx:filterExpr>
                  </xqx:stepExpr>
                  <xqx:stepExpr>
                    <xqx:xpathAxis>descendant-or-self</xqx:xpathAxis>
                    <xqx:anyKindTest/>
                  </xqx:stepExpr>
                  <xqx:stepExpr>
                    <xqx:xpathAxis>child</xqx:xpathAxis>
                    <xqx:nameTest>book</xqx:nameTest>
                  </xqx:stepExpr>
                </xqx:pathExpr>
              </xqx:forExpr>
            </xqx:forClauseItem>
          </xqx:forClause>
          <xqx:whereClause>
            <xqx:andOp>
              <xqx:firstOperand>
                <xqx:equalOp>
                  <xqx:firstOperand>
                    <xqx:pathExpr>
                      <xqx:stepExpr>
                        <xqx:filterExpr>
                          <xqx:varRef>
                            <xqx:name>b</xqx:name>
                          </xqx:varRef>
                        </xqx:filterExpr>
                      </xqx:stepExpr>
                      <xqx:stepExpr>
                        <xqx:xpathAxis>child</xqx:xpathAxis>
                        <xqx:nameTest>publisher</xqx:nameTest>
                      </xqx:stepExpr>
                    </xqx:pathExpr>
                  </xqx:firstOperand>
                  <xqx:secondOperand>
                    <xqx:stringConstantExpr>
                      <xqx:value>Addison-Wesley</xqx:value>
                    </xqx:stringConstantExpr>
                  </xqx:secondOperand>
                </xqx:equalOp>
              </xqx:firstOperand>
              <xqx:secondOperand>
```

```
        <xqx:greaterThanOp>
          <xqx:firstOperand>
            <xqx:pathExpr>
              <xqx:stepExpr>
                <xqx:filterExpr>
                  <xqx:varRef>
                    <xqx:name>b</xqx:name>
                  </xqx:varRef>
                </xqx:filterExpr>
              </xqx:stepExpr>
              <xqx:stepExpr>
                <xqx:xpathAxis>attribute</xqx:xpathAxis>
                <xqx:nameTest>year</xqx:nameTest>
              </xqx:stepExpr>
            </xqx:pathExpr>
          </xqx:firstOperand>
          <xqx:secondOperand>
            <xqx:integerConstantExpr>
              <xqx:value>1991</xqx:value>
            </xqx:integerConstantExpr>
          </xqx:secondOperand>
        </xqx:greaterThanOp>
      </xqx:secondOperand>
    </xqx:andOp>
  </xqx:whereClause>
  <xqx:orderByClause>
    <xqx:orderBySpec>
      <xqx:orderByExpr>
        <xqx:pathExpr>
          <xqx:stepExpr>
            <xqx:filterExpr>
              <xqx:varRef>
                <xqx:name>b</xqx:name>
              </xqx:varRef>
            </xqx:filterExpr>
          </xqx:stepExpr>
          <xqx:stepExpr>
            <xqx:xpathAxis>child</xqx:xpathAxis>
            <xqx:nameTest>title</xqx:nameTest>
          </xqx:stepExpr>
        </xqx:pathExpr>
      </xqx:orderByExpr>
    </xqx:orderBySpec>
  </xqx:orderByClause>
  <xqx:returnClause>
```

```
            <xqx:elementConstructor>
              <xqx:tagName>book</xqx:tagName>
              <xqx:elementContent>
                <xqx:pathExpr>
                  <xqx:stepExpr>
                    <xqx:filterExpr>
                      <xqx:varRef>
                        <xqx:name>b</xqx:name>
                      </xqx:varRef>
                    </xqx:filterExpr>
                  </xqx:stepExpr>
                  <xqx:stepExpr>
                    <xqx:xpathAxis>attribute</xqx:xpathAxis>
                    <xqx:nameTest>year</xqx:nameTest>
                  </xqx:stepExpr>
                </xqx:pathExpr>
                <xqx:pathExpr>
                  <xqx:stepExpr>
                    <xqx:filterExpr>
                      <xqx:varRef>
                        <xqx:name>b</xqx:name>
                      </xqx:varRef>
                    </xqx:filterExpr>
                  </xqx:stepExpr>
                  <xqx:stepExpr>
                    <xqx:xpathAxis>child</xqx:xpathAxis>
                    <xqx:nameTest>title</xqx:nameTest>
                  </xqx:stepExpr>
                </xqx:pathExpr>
              </xqx:elementContent>
            </xqx:elementConstructor>
          </xqx:returnClause>
        </xqx:flworExpr>
      </xqx:elementContent>
    </xqx:elementConstructor>
  </xqx:queryBody>
 </xqx:mainModule>
</xqx:module>
```

3.3.3. Transformation of XQueryX Solution into XQuery

Application of the stylesheet in Appendix B – Transforming XQueryX to XQuery on page 56 to the XQueryX solution results in:

```
<bib>{
```

```
 for $b in
doc("http://bstore1.example.com/bib.xml")/descendant-or-self::node()/child::book
 where (($b/child::publisher = "Addison-Wesley") and ($b/attribute::year > 1991))
 order by $b/child::title
 return <book>{$b/attribute::year}{$b/child::title}</book>
}</bib>
```

3.4. Example 4

Here is Q8 from the [XML Query Use Cases], use case NS (Queries Using Namespaces): "Select all traders (either seller or high bidder) without negative comments."

3.4.1. XQuery solution in XQuery Use Cases:

```
declare namespace ma = "http://www.example.com/AuctionWatch";
<Q8 xmlns:ma="http://www.example.com/AuctionWatch"
    xmlns:eachbay="http://www.example.com/auctioneers#eachbay"
    xmlns:xlink="http://www.w3.org/1999/xlink">
  {
    for $s in doc("auction.xml")//ma:Trading_Partners/(ma:Seller | ma:High_Bidder)
    where $s/*:NegativeComments = 0
    return $s
  }
</Q8>
```

3.4.2. A solution in XQueryX:

```
<?xml version="1.0"?>
<xqx:module xmlns:xqx="http://www.w3.org/2005/XQueryX"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://www.w3.org/2005/XQueryX
                        http://www.w3.org/2005/XQueryX/xqueryx.xsd">
  <xqx:mainModule>
    <xqx:prolog>
      <xqx:namespaceDecl>
        <xqx:prefix>ma</xqx:prefix>
        <xqx:uri>http://www.example.com/AuctionWatch</xqx:uri>
      </xqx:namespaceDecl>
    </xqx:prolog>
    <xqx:queryBody>
      <xqx:elementConstructor>
        <xqx:tagName>Q8</xqx:tagName>
        <xqx:attributeList>
          <xqx:namespaceDeclaration>
            <xqx:prefix>ma</xqx:prefix>
            <xqx:uri>http://www.example.com/AuctionWatch</xqx:uri>
          </xqx:namespaceDeclaration>
```

```
    <xqx:namespaceDeclaration>
      <xqx:prefix>eachbay</xqx:prefix>
      <xqx:uri>http://www.example.com/auctioneers#eachbay</xqx:uri>
    </xqx:namespaceDeclaration>
    <xqx:namespaceDeclaration>
      <xqx:prefix>xlink</xqx:prefix>
      <xqx:uri>http://www.w3.org/1999/xlink</xqx:uri>
    </xqx:namespaceDeclaration>
  </xqx:attributeList>
  <xqx:elementContent>
    <xqx:flworExpr>
      <xqx:forClause>
        <xqx:forClauseItem>
          <xqx:typedVariableBinding>
            <xqx:varName>s</xqx:varName>
          </xqx:typedVariableBinding>
          <xqx:forExpr>
            <xqx:pathExpr>
              <xqx:stepExpr>
                <xqx:filterExpr>
                  <xqx:functionCallExpr>
                    <xqx:functionName xqx:prefix="fn">doc</xqx:functionName>
                    <xqx:arguments>
                      <xqx:stringConstantExpr>
                        <xqx:value>auction.xml</xqx:value>
                      </xqx:stringConstantExpr>
                    </xqx:arguments>
                  </xqx:functionCallExpr>
                </xqx:filterExpr>
              </xqx:stepExpr>
              <xqx:stepExpr>
                <xqx:xpathAxis>descendant-or-self</xqx:xpathAxis>
                <xqx:anyKindTest/>
              </xqx:stepExpr>
              <xqx:stepExpr>
                <xqx:xpathAxis>child</xqx:xpathAxis>
                <xqx:nameTest xqx:prefix="ma">Trading_Partners</xqx:nameTest>

              </xqx:stepExpr>
              <xqx:stepExpr>
                <xqx:filterExpr>
                  <xqx:parenthesizedExpr>
                    <xqx:unionOp>
                      <xqx:firstOperand>
                        <xqx:pathExpr>
```

```
                                <xqx:stepExpr>
                                  <xqx:xpathAxis>child</xqx:xpathAxis>
                                 <xqx:nameTest xqx:prefix="ma">Seller</xqx:nameTest>

                                </xqx:stepExpr>
                              </xqx:pathExpr>
                            </xqx:firstOperand>
                            <xqx:secondOperand>
                              <xqx:pathExpr>
                                <xqx:stepExpr>
                                  <xqx:xpathAxis>child</xqx:xpathAxis>
                                  <xqx:nameTest
xqx:prefix="ma">High_Bidder</xqx:nameTest>
                                </xqx:stepExpr>
                              </xqx:pathExpr>
                            </xqx:secondOperand>
                          </xqx:unionOp>
                        </xqx:parenthesizedExpr>
                      </xqx:filterExpr>
                    </xqx:stepExpr>
                  </xqx:pathExpr>
                </xqx:forExpr>
              </xqx:forClauseItem>
            </xqx:forClause>
            <xqx:whereClause>
              <xqx:equalOp>
                <xqx:firstOperand>
                  <xqx:pathExpr>
                    <xqx:stepExpr>
                      <xqx:filterExpr>
                        <xqx:varRef>
                          <xqx:name>s</xqx:name>
                        </xqx:varRef>
                      </xqx:filterExpr>
                    </xqx:stepExpr>
                    <xqx:stepExpr>
                      <xqx:xpathAxis>child</xqx:xpathAxis>
                      <xqx:Wildcard>
                        <xqx:star/>
                        <xqx:NCName>NegativeComments</xqx:NCName>
                      </xqx:Wildcard>
                    </xqx:stepExpr>
                  </xqx:pathExpr>
                </xqx:firstOperand>
                <xqx:secondOperand>
```

```
                <xqx:integerConstantExpr>
                  <xqx:value>0</xqx:value>
                </xqx:integerConstantExpr>
              </xqx:secondOperand>
            </xqx:equalOp>
          </xqx:whereClause>
          <xqx:returnClause>
            <xqx:varRef>
              <xqx:name>s</xqx:name>
            </xqx:varRef>
          </xqx:returnClause>
        </xqx:flworExpr>
      </xqx:elementContent>
    </xqx:elementConstructor>
  </xqx:queryBody>
 </xqx:mainModule>
</xqx:module>
```

3.4.3. Transformation of XQueryX Solution into XQuery

Application of the stylesheet in Appendix B – Transforming XQueryX to XQuery on page 56 to the XQueryX solution results in:

```
declare namespace ma="http://www.example.com/AuctionWatch";
<Q8 xmlns:ma="http://www.example.com/AuctionWatch"
xmlns:eachbay="http://www.example.com/auctioneers#eachbay"
xmlns:xlink="http://www.w3.org/1999/xlink">{
 for $s in
fn:doc("auction.xml")/descendant-or-self::node()/child::ma:Trading_Partners/((child::ma:Seller
 union child::ma:High_Bidder))
 where ($s/child::*:NegativeComments = 0)
 return $s
}</Q8>
```

4. An XML Schema for the XQuery XML Syntax

Here is the XML Schema against which XQueryX documents must be valid.

```
<?xml version="1.0"?>
<!--
  Changes from CR document:
  * revised the text describing the trivial embedding feature (bug #2611)
  * refined the content model for xqx:stepExpr and xqx:pathExpr (bug #2523)
  * fixed the handling of an empty xqx:Wildcard element (bug #3022)
  * removed the unnecessary <xqx:parenthesizedExpr/> element (#3333)
```

```
  * always generated braces associated with a computed PI constructor value (bug
#3442)
  * escape/quote "<" and whitespace chars other than #x20 (bugs #3446 and #3474)
  * support schemaImport and moduleImport with multiple <xqx:targetLocation>s (bug
#3520)
  * restructure prolog details in preparation for future XQuery features
-->
<!--  Readers of this schema are reminded that the default value for both
      minOccurs and maxOccurs is "1". -->
 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             xmlns="http://www.w3.org/2005/XQueryX"
             targetNamespace="http://www.w3.org/2005/XQueryX"
             elementFormDefault="qualified" attributeFormDefault="qualified">


  <!-- A few helper declarations -->
  <xsd:complexType name="emptyContent"/>

  <xsd:element name="NCName" type="xsd:NCName"/>

  <xsd:complexType name="QName">
    <xsd:simpleContent>
      <xsd:extension base="xsd:NCName">
        <xsd:attribute name="prefix" type="xsd:NCName" use="optional"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>

  <!-- The base class -->
  <xsd:complexType name="expr"/>

  <xsd:element name="expr" type="expr" abstract="true"/>

  <!-- A list of expressions -->
  <xsd:complexType name="exprList">
    <xsd:sequence>
      <xsd:element ref="expr" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="exprWrapperOptional">
    <xsd:sequence>
      <xsd:element ref="expr" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
```

```
  <!-- Simple wrapper class -->
  <xsd:complexType name="exprWrapper">
    <xsd:sequence>
      <xsd:element ref="expr"/>
    </xsd:sequence>
  </xsd:complexType>

  <!-- constant expressions. We have 4 different subclasses for this -->
  <xsd:complexType name="constantExpr">
    <xsd:complexContent>
      <xsd:extension base="expr">
        <xsd:sequence>
          <xsd:element name="value" type="xsd:anyType"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="constantExpr" type="constantExpr" abstract="true"
substitutionGroup="expr"/>

  <xsd:complexType name="integerConstantExpr">
    <xsd:complexContent>
      <xsd:restriction base="constantExpr">
        <xsd:sequence>
          <xsd:element name="value" type="xsd:integer"/>
        </xsd:sequence>
      </xsd:restriction>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="integerConstantExpr" type="integerConstantExpr"
substitutionGroup="constantExpr"/>

  <xsd:complexType name="decimalConstantExpr">
    <xsd:complexContent>
      <xsd:restriction base="constantExpr">
        <xsd:sequence>
          <xsd:element name="value" type="xsd:decimal"/>
        </xsd:sequence>
      </xsd:restriction>
    </xsd:complexContent>
  </xsd:complexType>
```

```
  <xsd:element name="decimalConstantExpr" type="decimalConstantExpr"
substitutionGroup="constantExpr"/>

  <xsd:complexType name="doubleConstantExpr">
    <xsd:complexContent>
      <xsd:restriction base="constantExpr">
        <xsd:sequence>
          <xsd:element name="value" type="xsd:double"/>
        </xsd:sequence>
      </xsd:restriction>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="doubleConstantExpr" type="doubleConstantExpr"
substitutionGroup="constantExpr"/>

  <xsd:complexType name="stringConstantExpr">
    <xsd:complexContent>
      <xsd:restriction base="constantExpr">
        <xsd:sequence>
          <xsd:element name="value" type="xsd:string"/>
        </xsd:sequence>
      </xsd:restriction>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="stringConstantExpr" type="stringConstantExpr"
substitutionGroup="constantExpr"/>

  <!-- Variables -->
  <xsd:complexType name="varRef">
    <xsd:complexContent>
      <xsd:extension base="expr">
        <xsd:sequence>
          <xsd:element name="name" type="QName"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="varRef" type="varRef" substitutionGroup="expr"/>

  <!-- root and context-item expressions -->
  <!-- rootExpr deleted per Bugzilla Bug #2523 -->
  <xsd:complexType name="contextItemExpr">
```

```
    <xsd:complexContent>
      <xsd:extension base="expr"/>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="contextItemExpr" type="contextItemExpr"
substitutionGroup="expr"/>

  <!-- Pragmas and extension expressions -->
  <xsd:element name="pragma">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="pragmaName" type="QName"/>
        <xsd:element name="pragmaContents" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:complexType name="extensionExpr">
    <xsd:complexContent>
      <xsd:extension base="expr">
        <xsd:sequence>
          <xsd:element ref="pragma" maxOccurs="unbounded"/>
          <xsd:element name="argExpr" type="exprWrapper" minOccurs="0"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="extensionExpr" type="extensionExpr" substitutionGroup="expr"/>


  <xsd:complexType name="functionCallExpr">
    <xsd:complexContent>
      <xsd:extension base="expr">
        <xsd:sequence>
          <xsd:element name="functionName" type="QName"/>
          <xsd:element name="arguments" type="exprList" minOccurs="0"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="functionCallExpr" type="functionCallExpr"
substitutionGroup="expr"/>
```

```
  <!-- Constructor functions -->
  <xsd:complexType name="constructorFunctionExpr">
    <xsd:complexContent>
      <xsd:extension base="expr">
        <xsd:sequence>
          <xsd:element name="typeName" type="QName"/>
          <xsd:element name="argExpr" type="exprWrapper"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="constructorFunctionExpr" type="constructorFunctionExpr"
substitutionGroup="expr"/>

  <xsd:complexType name="sequenceExpr">
    <xsd:complexContent>
      <xsd:extension base="expr">
        <xsd:sequence>
          <xsd:element ref="expr" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="sequenceExpr" type="sequenceExpr" substitutionGroup="expr"/>

  <xsd:complexType name="rangeSequenceExpr">
    <xsd:complexContent>
      <xsd:extension base="expr">
        <xsd:sequence>
          <xsd:element name="startExpr" type="exprWrapper"/>
          <xsd:element name="endExpr" type="exprWrapper"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="rangeSequenceExpr" type="rangeSequenceExpr"
substitutionGroup="expr"/>

  <!-- Builtin operator expressions -->
  <xsd:complexType name="operatorExpr">
    <xsd:complexContent>
```

```
      <xsd:extension base="expr"/>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="unaryOperatorExpr">
    <xsd:complexContent>
      <xsd:extension base="operatorExpr">
        <xsd:sequence>
          <xsd:element name="operand" type="exprWrapper"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name="binaryOperatorExpr">
    <xsd:complexContent>
      <xsd:extension base="operatorExpr">
        <xsd:sequence>
          <xsd:element name="firstOperand" type="exprWrapper"/>
          <xsd:element name="secondOperand" type="exprWrapper"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="operatorExpr" type="operatorExpr" abstract="true"
substitutionGroup="expr"/>

  <xsd:element name="arithmeticOp" type="operatorExpr" abstract="true"
substitutionGroup="operatorExpr"/>

  <xsd:element name="addOp" type="binaryOperatorExpr"
substitutionGroup="arithmeticOp"/>

  <xsd:element name="subtractOp" type="binaryOperatorExpr"
substitutionGroup="arithmeticOp"/>

  <xsd:element name="multiplyOp" type="binaryOperatorExpr"
substitutionGroup="arithmeticOp"/>

  <xsd:element name="divOp" type="binaryOperatorExpr"
substitutionGroup="arithmeticOp"/>

  <xsd:element name="idivOp" type="binaryOperatorExpr"
substitutionGroup="arithmeticOp"/>
```

```
  <xsd:element name="modOp" type="binaryOperatorExpr"
substitutionGroup="arithmeticOp"/>

  <xsd:element name="unaryMinusOp" type="unaryOperatorExpr"
substitutionGroup="arithmeticOp"/>

  <xsd:element name="unaryPlusOp" type="unaryOperatorExpr"
substitutionGroup="arithmeticOp"/>

  <xsd:element name="comparisonOp" type="binaryOperatorExpr" abstract="true"
               substitutionGroup="operatorExpr"/>

  <xsd:element name="valueComparisonOp" type="binaryOperatorExpr" abstract="true"
               substitutionGroup="operatorExpr"/>

  <xsd:element name="eqOp" type="binaryOperatorExpr"
substitutionGroup="valueComparisonOp"/>

  <xsd:element name="neOp" type="binaryOperatorExpr"
substitutionGroup="valueComparisonOp"/>

  <xsd:element name="gtOp" type="binaryOperatorExpr"
substitutionGroup="valueComparisonOp"/>

  <xsd:element name="geOp" type="binaryOperatorExpr"
substitutionGroup="valueComparisonOp"/>

  <xsd:element name="ltOp" type="binaryOperatorExpr"
substitutionGroup="valueComparisonOp"/>

  <xsd:element name="leOp" type="binaryOperatorExpr"
substitutionGroup="valueComparisonOp"/>

  <xsd:element name="generalComparisonOp" type="binaryOperatorExpr" abstract="true"

               substitutionGroup="operatorExpr"/>

  <xsd:element name="equalOp" type="binaryOperatorExpr"
substitutionGroup="generalComparisonOp"/>

  <xsd:element name="notEqualOp" type="binaryOperatorExpr"
substitutionGroup="generalComparisonOp"/>

  <xsd:element name="lessThanOp" type="binaryOperatorExpr"
```

```
substitutionGroup="generalComparisonOp"/>

  <xsd:element name="lessThanOrEqualOp" type="binaryOperatorExpr"
               substitutionGroup="generalComparisonOp"/>

  <xsd:element name="greaterThanOp" type="binaryOperatorExpr"
substitutionGroup="generalComparisonOp"/>

  <xsd:element name="greaterThanOrEqualOp" type="binaryOperatorExpr"
               substitutionGroup="generalComparisonOp"/>

  <xsd:element name="nodeComparisonOp" type="binaryOperatorExpr" abstract="true"
               substitutionGroup="operatorExpr"/>

  <xsd:element name="isOp" type="binaryOperatorExpr"
substitutionGroup="nodeComparisonOp"/>

  <xsd:element name="orderComparisonOp" type="binaryOperatorExpr" abstract="true"
               substitutionGroup="operatorExpr"/>

  <xsd:element name="nodeBeforeOp" type="binaryOperatorExpr"
substitutionGroup="orderComparisonOp"/>

  <xsd:element name="nodeAfterOp" type="binaryOperatorExpr"
substitutionGroup="orderComparisonOp"/>

  <xsd:element name="logicalOp" type="binaryOperatorExpr" abstract="true"
               substitutionGroup="operatorExpr"/>

  <xsd:element name="andOp" type="binaryOperatorExpr" substitutionGroup="logicalOp"/>


  <xsd:element name="orOp" type="binaryOperatorExpr" substitutionGroup="logicalOp"/>


  <xsd:element name="setOp" type="binaryOperatorExpr" abstract="true"
substitutionGroup="operatorExpr"/>

  <xsd:element name="unionOp" type="binaryOperatorExpr" substitutionGroup="setOp"/>


  <xsd:element name="intersectOp" type="binaryOperatorExpr"
substitutionGroup="setOp"/>

  <xsd:element name="exceptOp" type="binaryOperatorExpr" substitutionGroup="setOp"/>
```

```
  <!-- Basic typenames -->
  <xsd:element name="atomicType" type="QName" substitutionGroup="itemType"/>

  <!-- Used in castable expression -->
  <xsd:element name="singleType">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="atomicType"/>
        <xsd:element name="optional" type="emptyContent" minOccurs="0"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="itemType" abstract="true"/>

  <xsd:complexType name="emptyItemTypeContent"/>

  <xsd:element name="anyItemType" type="emptyItemTypeContent"
substitutionGroup="itemType"/>

  <xsd:simpleType name="occurrenceIndicator">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="?"/>
      <xsd:enumeration value="*"/>
      <xsd:enumeration value="+"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:complexType name="sequenceType">
    <xsd:choice>
      <xsd:element name="voidSequenceType" type="emptyContent"/>
      <xsd:sequence>
        <xsd:element ref="itemType"/>
        <xsd:element name="occurrenceIndicator" type="occurrenceIndicator"
minOccurs="0"/>
      </xsd:sequence>
    </xsd:choice>
  </xsd:complexType>

  <xsd:element name="sequenceType" type="sequenceType"/>

  <xsd:element name="typeDeclaration" type="sequenceType"/>
```

```
  <!-- Represents a "typed" variable (for clause, let clause etc) -->
  <xsd:element name="typedVariableBinding">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="varName" type="QName"/>
        <xsd:element ref="typeDeclaration" minOccurs="0"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <!-- Represents an untyped variable for the "at" clause in a for clause -->
  <xsd:element name="positionalVariableBinding" type="QName"/>
<!--               substitutionGroup="FLWORVariableBindings"/> -->

  <xsd:element name="variableBinding" type="QName"/>

  <!-- Represents all variable bindings in a for or let clause except typed and
positional variable bindings -->
  <xsd:element name="forLetClauseItemExtensions" abstract="true"/>

  <xsd:complexType name="forClauseItem">
    <xsd:sequence>
   <xsd:element ref="typedVariableBinding"/>
   <xsd:element ref="positionalVariableBinding" minOccurs="0" maxOccurs="1"/>
      <xsd:element ref="forLetClauseItemExtensions" minOccurs="0"
maxOccurs="unbounded"/>
      <xsd:element name="forExpr" type="exprWrapper"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:element name="forClauseItem" type="forClauseItem"/>

  <xsd:element name="forClause">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="forClauseItem" minOccurs="1" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>


  <xsd:element name="letClauseItem">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:choice>
```

```
    <xsd:sequence>
            <xsd:element ref="typedVariableBinding"/>
   <xsd:element ref="forLetClauseItemExtensions" minOccurs="0" maxOccurs="unbounded"/>

    </xsd:sequence>
          <xsd:element ref="forLetClauseItemExtensions" minOccurs="0"
maxOccurs="unbounded"/>
  </xsd:choice>
        <xsd:element name="letExpr" type="exprWrapper"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="letClause">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="letClauseItem" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="whereClause" type="exprWrapper"/>

  <xsd:element name="returnClause" type="exprWrapper"/>

  <xsd:simpleType name="emptyOrderingMode">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="empty greatest"/>
      <xsd:enumeration value="empty least"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="orderingKind">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="ascending"/>
      <xsd:enumeration value="descending"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:element name="orderModifier">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="orderingKind" type="orderingKind" minOccurs="0"/>
       <xsd:element name="emptyOrderingMode" type="emptyOrderingMode" minOccurs="0"/>
```

```
      <xsd:element name="collation" type="xsd:string" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="orderBySpec">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="orderByExpr" type="exprWrapper"/>
      <xsd:element ref="orderModifier" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="orderByClause">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="stable" type="emptyContent" minOccurs="0"/>
      <xsd:element ref="orderBySpec" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<!-- This is the flwor expression -->
<xsd:complexType name="flworExpr">
  <xsd:complexContent>
    <xsd:extension base="expr">
      <xsd:sequence>
        <xsd:choice maxOccurs="unbounded">
          <xsd:element ref="forClause"/>
          <xsd:element ref="letClause"/>
        </xsd:choice>
        <xsd:element ref="whereClause" minOccurs="0"/>
        <xsd:element ref="orderByClause" minOccurs="0"/>
        <xsd:element ref="returnClause"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="flworExpr" type="flworExpr" substitutionGroup="expr"/>

<!-- conditional expressions -->
<xsd:complexType name="ifThenElseExpr">
  <xsd:complexContent>
```

```
    <xsd:extension base="expr">
      <xsd:sequence>
        <xsd:element name="ifClause" type="exprWrapper"/>
        <xsd:element name="thenClause" type="exprWrapper"/>
        <xsd:element name="elseClause" type="exprWrapper"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="ifThenElseExpr" type="ifThenElseExpr" substitutionGroup="expr"/>


<!-- The following clauses describe quantified expressions -->
<xsd:simpleType name="quantifier">
  <xsd:restriction base="xsd:NMTOKEN">
    <xsd:enumeration value="some"/>
    <xsd:enumeration value="every"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:element name="quantifiedExprInClause">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="typedVariableBinding"/>
      <xsd:element name="sourceExpr" type="exprWrapper"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:complexType name="quantifiedExpr">
  <xsd:complexContent>
    <xsd:extension base="expr">
      <xsd:sequence>
        <xsd:element name="quantifier" type="quantifier"/>
        <xsd:element ref="quantifiedExprInClause" maxOccurs="unbounded"/>
        <xsd:element name="predicateExpr" type="exprWrapper"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="quantifiedExpr" type="quantifiedExpr" substitutionGroup="expr"/>
```

```
<!-- handle the typeswitch construct -->
<!-- Note: no substitutionGroup as we cannot use this anywhere -->
<xsd:element name="typeswitchExprCaseClause">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="expr">
        <xsd:sequence>
          <xsd:element ref="variableBinding" minOccurs="0"/>
          <xsd:element ref="sequenceType"/>
          <xsd:element name="resultExpr" type="exprWrapper"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>

<!-- Note: no substitutionGroup as we cannot use this anywhere -->
<xsd:element name="typeswitchExprDefaultClause">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="expr">
        <xsd:sequence>
          <xsd:element ref="variableBinding" minOccurs="0"/>
          <xsd:element name="resultExpr" type="exprWrapper"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>

<xsd:complexType name="typeswitchExpr">
  <xsd:complexContent>
    <xsd:extension base="expr">
      <xsd:sequence>
        <xsd:element name="argExpr" type="exprWrapper"/>
        <xsd:element ref="typeswitchExprCaseClause" maxOccurs="unbounded"/>
        <xsd:element ref="typeswitchExprDefaultClause"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="typeswitchExpr" type="typeswitchExpr" substitutionGroup="expr"/>
```

```
<!-- instance-of expressions -->
<xsd:complexType name="instanceOfExpr">
  <xsd:complexContent>
    <xsd:extension base="expr">
      <xsd:sequence>
        <xsd:element name="argExpr" type="exprWrapper"/>
        <xsd:element ref="sequenceType"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="instanceOfExpr" type="instanceOfExpr" substitutionGroup="expr"/>


<!-- treat-as expressions -->
<xsd:complexType name="treatExpr">
  <xsd:complexContent>
    <xsd:extension base="expr">
      <xsd:sequence>
        <xsd:element name="argExpr" type="exprWrapper"/>
        <xsd:element ref="sequenceType"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="treatExpr" type="treatExpr" substitutionGroup="expr"/>

<!-- castable and cast expressions -->
<xsd:complexType name="castableExpr">
  <xsd:complexContent>
    <xsd:extension base="expr">
      <xsd:sequence>
        <xsd:element name="argExpr" type="exprWrapper"/>
        <xsd:element ref="singleType"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="castableExpr" type="castableExpr" substitutionGroup="expr"/>

<xsd:complexType name="castExpr">
  <xsd:complexContent>
```

```
      <xsd:extension base="expr">
        <xsd:sequence>
          <xsd:element name="argExpr" type="exprWrapper"/>
          <xsd:element ref="singleType"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="castExpr" type="castExpr" substitutionGroup="expr"/>

  <!-- Validate expressions -->
  <xsd:simpleType name="validationMode">
    <xsd:restriction base="xsd:NMTOKEN">
      <xsd:enumeration value="lax"/>
      <xsd:enumeration value="strict"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:complexType name="validateExpr">
    <xsd:complexContent>
      <xsd:extension base="expr">
        <xsd:sequence>
          <xsd:element name="validationMode" type="validationMode" minOccurs="0"/>
          <xsd:element name="argExpr" type="exprWrapper"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="validateExpr" type="validateExpr" substitutionGroup="expr"/>

  <!-- Direct constructors. Only elementConstructor for now -->
  <!-- Note the absence of constructors corresponding to
       the directCommentConstructor and the directPIConstructor
       productions in the XQuery grammar. This is because they are
       trivially identical to the computed variants
    -->

  <!-- attributeConstructor is no longer a subclass of expr -->
  <xsd:complexType name="attributeConstructor">
    <xsd:sequence>
      <xsd:element name="attributeName" type="QName"/>
      <xsd:choice>
        <xsd:element name="attributeValueExpr" type="exprList"/>
```

```
        <xsd:element name="attributeValue" type="xsd:string"/>
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="namespaceDeclaration">
    <xsd:sequence>
      <xsd:element name="prefix" type="xsd:NCName" minOccurs="0"/>
      <xsd:element name="uri" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:element name="attributeList">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:choice maxOccurs="unbounded">
          <xsd:element name="attributeConstructor" type="attributeConstructor"/>
          <xsd:element name="namespaceDeclaration" type="namespaceDeclaration"/>
        </xsd:choice>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="elementContent" type="exprList"/>

  <xsd:complexType name="elementConstructor">
    <xsd:complexContent>
      <xsd:extension base="expr">
        <xsd:sequence>
          <xsd:element name="tagName" type="QName"/>
          <xsd:element ref="attributeList" minOccurs="0"/>
          <xsd:element ref="elementContent" minOccurs="0"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="elementConstructor" type="elementConstructor"
substitutionGroup="expr"/>

  <!-- computed constructors -->
  <xsd:complexType name="computedElementConstructor">
    <xsd:complexContent>
      <xsd:extension base="expr">
        <xsd:sequence>
```

```
        <xsd:choice>
          <xsd:element name="tagName" type="QName"/>
          <xsd:element name="tagNameExpr" type="exprWrapper"/>
        </xsd:choice>
        <xsd:element name="contentExpr" type="exprWrapper" minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="computedElementConstructor" type="computedElementConstructor"
             substitutionGroup="expr"/>

<xsd:complexType name="computedAttributeConstructor">
  <xsd:complexContent>
    <xsd:extension base="expr">
      <xsd:sequence>
        <xsd:choice>
          <xsd:element name="tagName" type="QName"/>
          <xsd:element name="tagNameExpr" type="exprWrapper"/>
        </xsd:choice>
        <xsd:element name="valueExpr" type="exprWrapper" minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="computedAttributeConstructor" type="computedAttributeConstructor"

             substitutionGroup="expr"/>

<xsd:complexType name="computedDocumentConstructor">
  <xsd:complexContent>
    <xsd:extension base="expr">
      <xsd:sequence>
        <xsd:element name="argExpr" type="exprWrapper"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="computedDocumentConstructor" type="computedDocumentConstructor"

             substitutionGroup="expr"/>
```

```
  <xsd:complexType name="computedTextConstructor">
    <xsd:complexContent>
      <xsd:extension base="expr">
        <xsd:sequence>
          <xsd:element name="argExpr" type="exprWrapper" minOccurs="0"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="computedTextConstructor" type="computedTextConstructor"
substitutionGroup="expr"/>

  <xsd:complexType name="computedCommentConstructor">
    <xsd:complexContent>
      <xsd:extension base="expr">
        <xsd:sequence>
          <xsd:element name="argExpr" type="exprWrapper"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="computedCommentConstructor" type="computedCommentConstructor"
              substitutionGroup="expr"/>

  <xsd:complexType name="computedPIConstructor">
    <xsd:complexContent>
      <xsd:extension base="expr">
        <xsd:sequence>
          <xsd:choice>
            <xsd:element name="piTarget" type="xsd:NCName"/>
            <xsd:element name="piTargetExpr" type="exprWrapper"/>
          </xsd:choice>
          <xsd:element name="piValueExpr" type="exprWrapper" minOccurs="0"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="computedPIConstructor" type="computedPIConstructor"
substitutionGroup="expr"/>

  <!-- ordered and unordered expressions -->
  <xsd:complexType name="unorderedExpr">
```

```
    <xsd:complexContent>
      <xsd:extension base="expr">
        <xsd:sequence>
          <xsd:element name="argExpr" type="exprWrapper"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="unorderedExpr" type="unorderedExpr" substitutionGroup="expr"/>


  <xsd:complexType name="orderedExpr">
    <xsd:complexContent>
      <xsd:extension base="expr">
        <xsd:sequence>
          <xsd:element name="argExpr" type="exprWrapper"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="orderedExpr" type="orderedExpr" substitutionGroup="expr"/>

  <xsd:complexType name="simpleWildcard">
    <xsd:choice>
      <xsd:element name="QName" type="QName"/>
      <xsd:element name="star" type="emptyContent"/>
    </xsd:choice>
  </xsd:complexType>

  <xsd:element name="Wildcard">
    <xsd:complexType>
      <xsd:all>
        <xsd:element name="star" type="emptyContent" minOccurs="0"/>
        <xsd:element ref="NCName" minOccurs="0"/>
      </xsd:all>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="schemaAttributeTest" type="QName" substitutionGroup="kindTest"/>


  <xsd:element name="attributeTest" substitutionGroup="kindTest">
    <xsd:complexType>
```

```
      <xsd:complexContent>
        <xsd:extension base="emptyItemTypeContent">
          <xsd:sequence minOccurs="0">
            <xsd:element name="attributeName" type="simpleWildcard"/>
            <xsd:element name="typeName" type="QName" minOccurs="0"/>
          </xsd:sequence>
        </xsd:extension>
      </xsd:complexContent>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="anyElementTest" abstract="true" substitutionGroup="kindTest"/>


  <xsd:element name="schemaElementTest" type="QName"
substitutionGroup="anyElementTest"/>

  <xsd:element name="elementTest" substitutionGroup="anyElementTest">
    <xsd:complexType>
      <xsd:complexContent>
        <xsd:extension base="emptyItemTypeContent">
          <xsd:sequence minOccurs="0">
            <xsd:element name="elementName" type="simpleWildcard"/>
            <xsd:sequence minOccurs="0">
              <xsd:element name="typeName" type="QName"/>
              <xsd:element name="nillable" type="emptyContent" minOccurs="0"/>
            </xsd:sequence>
          </xsd:sequence>
        </xsd:extension>
      </xsd:complexContent>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="documentTest" substitutionGroup="kindTest">
    <xsd:complexType>
      <xsd:complexContent>
        <xsd:extension base="emptyItemTypeContent">
          <xsd:sequence>
             <xsd:element ref="anyElementTest" minOccurs="0"/>
          </xsd:sequence>
        </xsd:extension>
      </xsd:complexContent>
    </xsd:complexType>
  </xsd:element>
```

```
  <xsd:element name="piTest" substitutionGroup="kindTest">
    <xsd:complexType>
      <xsd:complexContent>
        <xsd:extension base="emptyItemTypeContent">
          <xsd:sequence>
            <xsd:element name="piTarget" type="xsd:NCName" minOccurs="0"/>
          </xsd:sequence>
        </xsd:extension>
      </xsd:complexContent>
    </xsd:complexType>
  </xsd:element>


  <xsd:element name="nameTest" type="QName"/>

  <xsd:element name="kindTest" substitutionGroup="itemType"/>

  <xsd:element name="textTest" type="emptyItemTypeContent"
substitutionGroup="kindTest"/>

  <xsd:element name="commentTest" type="emptyItemTypeContent"
substitutionGroup="kindTest"/>

  <xsd:element name="anyKindTest" type="emptyItemTypeContent"
substitutionGroup="kindTest"/>


  <xsd:element name="xpathAxis">
    <xsd:simpleType>
      <xsd:restriction base="xsd:NMTOKEN">
        <xsd:enumeration value="child"/>
        <xsd:enumeration value="attribute"/>
        <xsd:enumeration value="self"/>
        <xsd:enumeration value="parent"/>
        <xsd:enumeration value="descendant-or-self"/>
        <xsd:enumeration value="descendant"/>
        <xsd:enumeration value="following"/>
        <xsd:enumeration value="following-sibling"/>
        <xsd:enumeration value="ancestor"/>
        <xsd:enumeration value="ancestor-or-self"/>
        <xsd:enumeration value="preceding"/>
        <xsd:enumeration value="preceding-sibling"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:element>
```

```
<!-- xqx:parenthesizedExpr deemed not useful; removed.
  <xsd:element name="parenthesizedExpr" type="exprWrapperOptional"/>
-->

  <xsd:group name="filterExpr">
    <xsd:choice>
      <xsd:element ref="constantExpr"/>
      <xsd:element ref="varRef"/>
      <xsd:element ref="contextItemExpr"/>
      <xsd:element ref="functionCallExpr"/>
<!-- xqx:parenthesizedExpr deemed not useful; replaced with xqx:sequenceExpr. -->
      <xsd:element ref="sequenceExpr"/>
      <xsd:element ref="elementConstructor"/>
      <xsd:element ref="computedElementConstructor"/>
      <xsd:element ref="computedAttributeConstructor"/>
      <xsd:element ref="computedDocumentConstructor"/>
      <xsd:element ref="computedTextConstructor"/>
      <xsd:element ref="computedCommentConstructor"/>
      <xsd:element ref="computedPIConstructor"/>
      <xsd:element ref="orderedExpr"/>
      <xsd:element ref="unorderedExpr"/>
    </xsd:choice>
  </xsd:group>

  <!-- removed nameTest and Wildcard outer choices per Bugzilla Bug #2523 -->
  <xsd:element name="stepExpr">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:choice>
          <xsd:sequence>
            <xsd:element ref="xpathAxis"/>
            <xsd:choice>
              <xsd:element ref="kindTest"/>
              <xsd:element ref="nameTest"/>
              <xsd:element ref="Wildcard"/>
              </xsd:choice>
          </xsd:sequence>
          <xsd:element name="filterExpr">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:group ref="filterExpr"/>
              </xsd:sequence>
            </xsd:complexType>
```

```
        </xsd:element>
      </xsd:choice>
      <xsd:element name="predicates" type="exprList" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<!-- rewrote pathExpr definition per Bugzilla Bug #2523 -->
<xsd:complexType name="pathExpr">
  <xsd:complexContent>
    <xsd:extension base="expr">
      <xsd:choice>
        <xsd:sequence>
          <xsd:element name="rootExpr" type="emptyContent"/>
          <xsd:element ref="stepExpr" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:element ref="stepExpr" maxOccurs="unbounded"/>
      </xsd:choice>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="pathExpr" type="pathExpr" substitutionGroup="expr"/>

<!-- The following constructs deal with the query prolog -->
<xsd:element name="module">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="versionDecl" minOccurs="0"/>
      <xsd:choice>
        <xsd:element ref="mainModule"/>
        <xsd:element ref="libraryModule"/>
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="mainModule">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="prolog" minOccurs="0"/>
      <xsd:element name="queryBody" type="exprWrapper"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

```
 <xsd:element name="libraryModule">
   <xsd:complexType>
     <xsd:sequence>
       <xsd:element ref="moduleDecl"/>
       <xsd:element ref="prolog" minOccurs="0"/>
     </xsd:sequence>
   </xsd:complexType>
 </xsd:element>

 <xsd:element name="versionDecl">
   <xsd:complexType>
     <xsd:sequence>
       <xsd:element name="version" type="xsd:string"/>
     </xsd:sequence>
   </xsd:complexType>
 </xsd:element>

 <xsd:element name="prolog">
   <xsd:complexType>
     <xsd:sequence>
    <xsd:element ref="prologPartOneItem" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="prologPartTwoItem" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
 </xsd:element>

 <xsd:element name="prologPartOneItem" abstract="true"/>

 <xsd:element name="prologPartTwoItem" abstract="true"/>

 <xsd:element name="defaultCollationDecl" type="xsd:string"
              substitutionGroup="prologPartOneItem"/>

 <xsd:element name="baseUriDecl" type="xsd:string"
              substitutionGroup="prologPartOneItem"/>

 <xsd:element name="constructionDecl" substitutionGroup="prologPartOneItem">
   <xsd:simpleType>
     <xsd:restriction base="xsd:NMTOKEN">
       <xsd:enumeration value="strip"/>
       <xsd:enumeration value="preserve"/>
     </xsd:restriction>
   </xsd:simpleType>
 </xsd:element>
```

```
<xsd:element name="orderingModeDecl" substitutionGroup="prologPartOneItem">
  <xsd:simpleType>
    <xsd:restriction base="xsd:NMTOKEN">
      <xsd:enumeration value="ordered"/>
      <xsd:enumeration value="unordered"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>

<xsd:element name="emptyOrderingDecl" type="emptyOrderingMode"
             substitutionGroup="prologPartOneItem"/>

<xsd:element name="copyNamespacesDecl" substitutionGroup="prologPartOneItem">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="preserveMode">
        <xsd:simpleType>
          <xsd:restriction base="xsd:NMTOKEN">
            <xsd:enumeration value="preserve"/>
            <xsd:enumeration value="no-preserve"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:element>
      <xsd:element name="inheritMode">
        <xsd:simpleType>
          <xsd:restriction base="xsd:NMTOKEN">
            <xsd:enumeration value="inherit"/>
            <xsd:enumeration value="no-inherit"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:simpleType name="defaultNamespaceCategory">
  <xsd:restriction base="xsd:NMTOKEN">
    <xsd:enumeration value="function"/>
    <xsd:enumeration value="element"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:element name="defaultNamespaceDecl" substitutionGroup="prologPartOneItem">
  <xsd:complexType>
```

```
      <xsd:sequence>
       <xsd:element name="defaultNamespaceCategory" type="defaultNamespaceCategory"/>

        <xsd:element name="uri" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:complexType name="namespaceDeclType">
    <xsd:sequence>
      <xsd:element name="prefix" type="xsd:NCName"/>
      <xsd:element name="uri" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:element name="namespaceDecl" type="namespaceDeclType"
               substitutionGroup="prologPartOneItem"/>

  <xsd:element name="moduleDecl" type="namespaceDeclType"/>

  <xsd:element name="schemaImport" substitutionGroup="prologPartOneItem">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:choice minOccurs="0">
          <xsd:element name="namespacePrefix" type="xsd:NCName"/>
          <xsd:element name="defaultElementNamespace" type="emptyContent"/>
        </xsd:choice>
        <xsd:element name="targetNamespace" type="xsd:string"/>
        <xsd:element name="targetLocation" type="xsd:string" minOccurs="0"
maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="moduleImport" substitutionGroup="prologPartOneItem">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="namespacePrefix" type="xsd:NCName" minOccurs="0"/>
        <xsd:element name="targetNamespace" type="xsd:string"/>
        <xsd:element name="targetLocation" type="xsd:string" minOccurs="0"
maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
```

```
<xsd:element name="varDecl" substitutionGroup="prologPartTwoItem">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="varName" type="QName"/>
      <xsd:element ref="typeDeclaration" minOccurs="0"/>
      <xsd:choice>
        <xsd:element name="varValue" type="exprWrapper"/>
        <xsd:element name="external" type="emptyContent"/>
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="optionDecl" substitutionGroup="prologPartTwoItem">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="optionName" type="QName"/>
      <xsd:element name="optionContents" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="functionDecl" substitutionGroup="prologPartTwoItem">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="functionName" type="QName"/>
      <xsd:element ref="paramList"/>
      <xsd:element ref="typeDeclaration" minOccurs="0"/>
      <xsd:choice>
        <xsd:element name="functionBody" type="exprWrapper"/>
        <xsd:element name="externalDefinition" type="emptyContent"/>
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>


<xsd:element name="param">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="varName" type="QName"/>
      <xsd:element ref="typeDeclaration" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

```
  <xsd:element name="paramList">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="param" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <!-- The element below handles the trivial XQuery embedding -->
  <xsd:element name="xquery" type="xsd:string"/>

</xsd:schema>
```

5. Conformance

This section defines the conformance criteria for an XQueryX processor (see Figure 1, "Processing Model Overview", in [XQuery 1.0: An XML Query Language], .

In this section, the following terms are used to indicate the requirement levels defined in [RFC 2119]. [Definition: MUST means that the item is an absolute requirement of the specification.] [Definition: SHOULD means that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.] [Definition: MAY means that an item is truly optional.]

An XQueryX processor that claims to conform to this specification MUST implement the XQueryX syntax as defined in § 4 – An XML Schema for the XQuery XML Syntax on page 26 of this specification and include a claim of Minimal Conformance as defined in [XQuery 1.0: An XML Query Language], . In addition to a claim of Minimal Conformance, it MAY claim conformance to one or more optional features defined in [XQuery 1.0: An XML Query Language], .

Appendix A. References

XML Query 1.0 Requirements

XQuery 1.0: An XML Query Language

XML Query Use Cases

XQuery 1.0 Formal Semantics

RFC 2119

S. Bradner. *Key Words for use in RFCs to Indicate Requirement Levels.* IETF RFC 2119. See http://www.ietf.org/rfc/rfc2119.txt.

RFC 3023

M. Murata, S. St. Laurent, and D. Kohn. *XML Media Types* IETF RFC 3023 or its successors. See http://www.ietf.org/rfc/rfc3023.txt.

Appendix B. Transforming XQueryX to XQuery

The following stylesheet converts from XQueryX syntax to XML Query syntax. Note the intent of the stylesheet is to produce *a* valid XQuery with the same semantics (see [XQuery 1.0 Formal Semantics]) as the input XQueryX.

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xqx="http://www.w3.org/2005/XQueryX">

<!-- Note that this stylesheet frequently invokes templates for
     specified elements, even when there are no templates in the
     stylesheet whose match="" attribute identifies those elements.
     In such case, the default template's element matching template
     is invoked, which merely invokes xsl:apply-templates -->

  <xsl:output method="text"/>
  <xsl:strip-space elements="*"/>
  <xsl:preserve-space elements="xqx:value xqx:attributeValue xqx:pragmaContents
                                xqx:optionContents xqx:xquery"/>

  <xsl:variable name="DOT" select="'.'"/>
  <xsl:variable name="SPACE" select="' '"/>
  <xsl:variable name="SLASH" select="'/'"/>
  <xsl:variable name="SLASH_SLASH" select="'//'"/>
  <xsl:variable name="LESSTHAN" select="'&lt;'"/>
  <xsl:variable name="GREATERTHAN" select="'&gt;'"/>
  <xsl:variable name="LPAREN" select="'('"/>
  <xsl:variable name="RPAREN" select="')'"/>
  <xsl:variable name="NEWLINE"><xsl:text>
</xsl:text></xsl:variable>
  <xsl:variable name="COMMA" select="','"/>
  <xsl:variable name="COMMA_SPACE" select="', '"/>
  <xsl:variable name="COMMA_NEWLINE"><xsl:text>,
</xsl:text></xsl:variable>
  <xsl:variable name="QUOTE"><xsl:text>'</xsl:text></xsl:variable>
  <xsl:variable name="DOUBLEQUOTE"><xsl:text>"</xsl:text></xsl:variable>
  <xsl:variable name="TO" select="' to '"/>
  <xsl:variable name="LBRACE" select="'{'"/>
  <xsl:variable name="RBRACE" select="'}'"/>
  <xsl:variable name="LBRACKET" select="'['"/>
  <xsl:variable name="RBRACKET" select="']'"/>
  <xsl:variable name="DOLLAR" select="'$'"/>
  <xsl:variable name="MINUS" select="'-'"/>
```

```
<xsl:variable name="PLUS" select="'+'"/>
<xsl:variable name="EQUAL" select="'='"/>
<xsl:variable name="COLON" select="':'"/>
<xsl:variable name="DOUBLE_COLON" select="'::'"/>
<xsl:variable name="SEMICOLON" select="';'"/>
<xsl:variable name="AT" select="'@'"/>
<xsl:variable name="STAR" select="'*'"/>
<xsl:variable name="QUESTIONMARK" select="'?'"/>
<xsl:variable name="ASSIGN" select="':='"/>
<xsl:variable name="SEPARATOR" select="';'"/>
<xsl:variable name="PRAGMA_BEGIN" select="'(# '"/>
<xsl:variable name="PRAGMA_END" select="' #)'"/>


<xsl:template name="delimitedList">
  <xsl:param name="delimiter" />
  <xsl:param name="leftEncloser"/>
  <xsl:param name="rightEncloser" />
  <xsl:param name="selector"/>

  <xsl:value-of select="$leftEncloser"/>
  <xsl:for-each select="*">
    <xsl:apply-templates select="."/>
    <xsl:if test="not (position()=last())">
      <xsl:value-of select="$delimiter"/>
    </xsl:if>
  </xsl:for-each>
  <xsl:value-of select="$rightEncloser"/>
</xsl:template>


<xsl:template name="parenthesizedList">
  <xsl:param name="delimiter" select="$COMMA_SPACE"/>
  <xsl:call-template name="delimitedList">
    <xsl:with-param name="delimiter" select="$delimiter" />
    <xsl:with-param name="leftEncloser" select="$LPAREN"/>
    <xsl:with-param name="rightEncloser" select="$RPAREN"/>
  </xsl:call-template>
</xsl:template>


<xsl:template name="commaSeparatedList">
  <xsl:call-template name="delimitedList">
    <xsl:with-param name="delimiter">
      <xsl:value-of select="$COMMA_SPACE"/>
```

```
      </xsl:with-param>
    </xsl:call-template>
  </xsl:template>


  <!-- To resolve Bugzilla bug #3446, we now escape CR (#xD), NEL (#x85),
       and LINE SEPARATOR (#x2028) characters in text nodes and attribute values.
         Note that this template is invoked for a number of other purposes (e.g.,
       xqx:collation, xqx:namespaceDeclaration) where the presence of such
       characters would be invalid and thus are highly unlikely to appear.
       If they do, then this template will happily escape them, deferring the
       error until the resulting XQuery expression is processed. -->
  <xsl:template name="quote">
    <xsl:param name="item"/>
    <xsl:value-of select="$DOUBLEQUOTE"/>
    <xsl:call-template name="globalReplace">
      <xsl:with-param name="stringToBeFixed">
        <xsl:call-template name="globalReplace">
          <xsl:with-param name="stringToBeFixed">
            <xsl:call-template name="globalReplace">
              <xsl:with-param name="stringToBeFixed">
                <xsl:call-template name="globalReplace">
                  <xsl:with-param name="stringToBeFixed">
                    <xsl:call-template name="globalReplace">
                      <xsl:with-param name="stringToBeFixed">
                        <xsl:call-template name="globalReplace">
                          <xsl:with-param name="stringToBeFixed">
                            <xsl:value-of select="$item"/>
                          </xsl:with-param>
                         <xsl:with-param name="toBeReplaced">&amp;</xsl:with-param>

                          <xsl:with-param
name="replacement">&amp;amp;</xsl:with-param>
                        </xsl:call-template>
          </xsl:with-param>
                      <xsl:with-param name="toBeReplaced">&lt;</xsl:with-param>
                      <xsl:with-param name="replacement">&amp;lt;</xsl:with-param>
                    </xsl:call-template>
                  </xsl:with-param>
                  <xsl:with-param name="toBeReplaced" select="'&#x85;'"/>
                  <xsl:with-param name="replacement">&amp;#x85;</xsl:with-param>
                </xsl:call-template>
              </xsl:with-param>
              <xsl:with-param name="toBeReplaced" select="'&#xD;'"/>
              <xsl:with-param name="replacement">&amp;#xD;</xsl:with-param>
```

```
            </xsl:call-template>
          </xsl:with-param>
          <xsl:with-param name="toBeReplaced" select="'&#x2028;'"/>
          <xsl:with-param name="replacement">&amp;#x2028;</xsl:with-param>
        </xsl:call-template>
      </xsl:with-param>
      <xsl:with-param name="toBeReplaced"><xsl:text>"</xsl:text></xsl:with-param>
      <xsl:with-param name="replacement"><xsl:text>""</xsl:text></xsl:with-param>
    </xsl:call-template>
    <xsl:value-of select="$DOUBLEQUOTE"/>
  </xsl:template>


  <xsl:template name="globalReplace">
    <xsl:param name="stringToBeFixed"/>
    <xsl:param name="toBeReplaced"/>
    <xsl:param name="replacement"/>
    <xsl:choose>
      <xsl:when test="contains($stringToBeFixed, $toBeReplaced)">
        <xsl:value-of select="concat(substring-before($stringToBeFixed,
$toBeReplaced), $replacement)"/>
        <xsl:call-template name="globalReplace">
          <xsl:with-param name="stringToBeFixed"
select="substring-after($stringToBeFixed, $toBeReplaced)"/>
          <xsl:with-param name="toBeReplaced" select="$toBeReplaced"/>
          <xsl:with-param name="replacement" select="$replacement"/>
        </xsl:call-template>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="$stringToBeFixed"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>


  <xsl:template match="xqx:QName | xqx:pragmaName | xqx:typeName | xqx:varName |
                       xqx:functionName | xqx:optionName |
                       xqx:atomicType | xqx:tagName">
    <xsl:if test="@xqx:prefix">
      <xsl:value-of select="@xqx:prefix"/>
      <xsl:value-of select="$COLON"/>
    </xsl:if>
    <xsl:value-of select="."/>
  </xsl:template>
```

```
  <xsl:template match="xqx:attributeName">
    <xsl:choose>
      <xsl:when test="@xqx:prefix='xmlns'">
        <xsl:message terminate="yes">Incorrect XQueryX: Attribute names are not
permitted to have prefix 'xmlns'; use xqx:namespaceDeclaration to declare
namespaces</xsl:message>
      </xsl:when>
      <xsl:when test="@xqx:prefix">
        <xsl:value-of select="@xqx:prefix"/>
        <xsl:value-of select="$COLON"/>
        <xsl:value-of select="."/>
      </xsl:when>
      <xsl:when test=". = 'xmlns'">
        <xsl:message terminate="yes">Incorrect XQueryX: Attribute names are not
permitted to be 'xmlns'; use xqx:namespaceDeclaration to declare
namespaces</xsl:message>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="."/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>


  <xsl:template match="xqx:NCName">
    <xsl:value-of select="."/>
  </xsl:template>


  <xsl:template match="xqx:rootExpr">
    <xsl:value-of select="$SLASH"/>
  </xsl:template>


  <xsl:template match="xqx:contextItemExpr">
    <xsl:value-of select="$DOT"/>
  </xsl:template>


  <xsl:template match="xqx:stringConstantExpr">
    <xsl:call-template name="quote">
      <xsl:with-param name="item" select="xqx:value"/>
    </xsl:call-template>
  </xsl:template>
```

```
  <xsl:template match="xqx:integerConstantExpr
      | xqx:decimalConstantExpr
      | xqx:doubleConstantExpr">
    <xsl:value-of select="xqx:value"/>
  </xsl:template>


  <xsl:template match="xqx:varRef">
    <xsl:value-of select="$DOLLAR"/>
    <xsl:if test="xqx:name/@xqx:prefix">
      <xsl:value-of select="xqx:name/@xqx:prefix"/>
      <xsl:value-of select="$COLON"/>
    </xsl:if>
    <xsl:value-of select="xqx:name"/>
  </xsl:template>


  <xsl:template match="xqx:pragma">
    <xsl:value-of select="$PRAGMA_BEGIN"/>
    <xsl:apply-templates select="xqx:pragmaName"/>
    <xsl:value-of select="$SPACE"/>
    <xsl:value-of select="xqx:pragmaContents"/>
    <xsl:value-of select="$PRAGMA_END"/>
  </xsl:template>


  <xsl:template match="xqx:extensionExpr">
    <xsl:apply-templates select="xqx:pragma"/>
    <xsl:value-of select="$LBRACE"/>
    <xsl:apply-templates select="xqx:argExpr"/>
    <xsl:value-of select="$RBRACE"/>
  </xsl:template>

<!-- Response to Bugzilla bug #2528 -->
   <xsl:template match="xqx:functionCallExpr">
     <xsl:choose>
       <xsl:when test="xqx:functionName/@xqx:prefix">
         <xsl:value-of select="xqx:functionName/@xqx:prefix"/>
         <xsl:value-of select="$COLON"/>
       </xsl:when>
       <xsl:otherwise>
         <xsl:if test="xqx:functionName = 'node' or
                       xqx:functionName = 'document-node' or
```

```
                        xqx:functionName = 'element' or
                        xqx:functionName = 'attribute' or
                        xqx:functionName = 'schema-element' or
                        xqx:functionName = 'schema-attribute' or
                        xqx:functionName = 'processing-instruction' or
                        xqx:functionName = 'comment' or
                        xqx:functionName = 'text' or
                        xqx:functionName = 'item' or
                        xqx:functionName = 'if' or
                        xqx:functionName = 'typeswitch' or
                        xqx:functionName = 'empty-sequence'">
          <xsl:variable name="message"><xsl:text>Incorrect XQueryX: function calls
 must not use unqualified "reserved" name "</xsl:text><xsl:value-of
select="xqx:functionName"/><xsl:text>"</xsl:text></xsl:variable>
           <xsl:message terminate="yes"><xsl:value-of
select="$message"/></xsl:message>
         </xsl:if>
       </xsl:otherwise>
     </xsl:choose>
     <xsl:value-of select="xqx:functionName"/>
     <xsl:choose>
       <xsl:when test="xqx:arguments">
         <xsl:for-each select="xqx:arguments">
           <xsl:call-template name="parenthesizedList"/>
         </xsl:for-each>
       </xsl:when>
       <xsl:otherwise>
         <xsl:value-of select="$LPAREN"/>
         <xsl:value-of select="$RPAREN"/>
       </xsl:otherwise>
     </xsl:choose>
   </xsl:template>


  <xsl:template match="xqx:constructorFunctionExpr">
    <xsl:apply-templates select="xqx:typeName"/>
    <xsl:for-each select="xqx:argExpr">
      <xsl:call-template name="parenthesizedList"/>
    </xsl:for-each>
  </xsl:template>


  <xsl:template match="xqx:unaryMinusOp | xqx:unaryPlusOp">
    <xsl:value-of select="$LPAREN"/>
     <xsl:choose>
```

```
      <xsl:when test="self::xqx:unaryPlusOp"><xsl:value-of
select="$PLUS"/></xsl:when>
      <xsl:when test="self::xqx:unaryMinusOp"><xsl:value-of
select="$MINUS"/></xsl:when>
     </xsl:choose>
    <xsl:apply-templates select="xqx:operand"/>
    <xsl:value-of select="$RPAREN"/>
  </xsl:template>


  <xsl:template match="xqx:addOp | xqx:subtractOp | xqx:multiplyOp
                     | xqx:divOp | xqx:idivOp | xqx:modOp">
    <xsl:value-of select="$LPAREN"/>
     <xsl:apply-templates select="xqx:firstOperand"/>
     <xsl:choose>
       <xsl:when test="self::xqx:addOp"><xsl:value-of select="$PLUS"/></xsl:when>
       <xsl:when test="self::xqx:subtractOp"><xsl:text> </xsl:text><xsl:value-of
select="$MINUS"/><xsl:text> </xsl:text></xsl:when>
      <xsl:when test="self::xqx:multiplyOp"><xsl:value-of select="$STAR"/></xsl:when>

       <xsl:when test="self::xqx:divOp"><xsl:text> div </xsl:text></xsl:when>
       <xsl:when test="self::xqx:idivOp"><xsl:text> idiv </xsl:text></xsl:when>
       <xsl:when test="self::xqx:modOp"><xsl:text> mod </xsl:text></xsl:when>
     </xsl:choose>
    <xsl:apply-templates select="xqx:secondOperand"/>
    <xsl:value-of select="$RPAREN"/>
  </xsl:template>


  <xsl:template match="xqx:eqOp | xqx:neOp | xqx:ltOp
                     | xqx:gtOp | xqx:leOp | xqx:geOp">
    <xsl:value-of select="$LPAREN"/>
     <xsl:apply-templates select="xqx:firstOperand"/>
     <xsl:choose>
       <xsl:when test="self::xqx:eqOp"><xsl:text> eq </xsl:text></xsl:when>
       <xsl:when test="self::xqx:neOp"><xsl:text> ne </xsl:text></xsl:when>
       <xsl:when test="self::xqx:ltOp"><xsl:text> lt </xsl:text></xsl:when>
       <xsl:when test="self::xqx:gtOp"><xsl:text> gt </xsl:text></xsl:when>
       <xsl:when test="self::xqx:leOp"><xsl:text> le </xsl:text></xsl:when>
       <xsl:when test="self::xqx:geOp"><xsl:text> ge </xsl:text></xsl:when>
     </xsl:choose>
    <xsl:apply-templates select="xqx:secondOperand"/>
    <xsl:value-of select="$RPAREN"/>
  </xsl:template>
```

```
  <xsl:template match="xqx:equalOp | xqx:notEqualOp | xqx:lessThanOp
                     | xqx:greaterThanOp | xqx:lessThanOrEqualOp |
xqx:greaterThanOrEqualOp">
    <xsl:value-of select="$LPAREN"/>
     <xsl:apply-templates select="xqx:firstOperand"/>
     <xsl:choose>
       <xsl:when test="self::xqx:equalOp">
        <xsl:text> </xsl:text><xsl:value-of select="$EQUAL"/><xsl:text> </xsl:text>

       </xsl:when>
       <xsl:when test="self::xqx:notEqualOp">
        <xsl:text> !</xsl:text><xsl:value-of select="$EQUAL"/><xsl:text> </xsl:text>

       </xsl:when>
       <xsl:when test="self::xqx:lessThanOp">
         <xsl:text> </xsl:text><xsl:value-of select="$LESSTHAN"/><xsl:text>
</xsl:text>
       </xsl:when>
       <xsl:when test="self::xqx:greaterThanOp">
         <xsl:text> </xsl:text><xsl:value-of select="$GREATERTHAN"/><xsl:text>
</xsl:text>
       </xsl:when>
       <xsl:when test="self::xqx:lessThanOrEqualOp">
         <xsl:text> </xsl:text><xsl:value-of select="$LESSTHAN"/>
         <xsl:value-of select="$EQUAL"/><xsl:text> </xsl:text>
       </xsl:when>
       <xsl:when test="self::xqx:greaterThanOrEqualOp">
         <xsl:text> </xsl:text><xsl:value-of select="$GREATERTHAN"/>
         <xsl:value-of select="$EQUAL"/><xsl:text> </xsl:text>
       </xsl:when>
     </xsl:choose>
    <xsl:apply-templates select="xqx:secondOperand"/>
    <xsl:value-of select="$RPAREN"/>
  </xsl:template>


  <xsl:template match="xqx:isOp | xqx:nodeBeforeOp | xqx:nodeAfterOp">
    <xsl:value-of select="$LPAREN"/>
     <xsl:apply-templates select="xqx:firstOperand"/>
     <xsl:choose>
       <xsl:when test="self::xqx:isOp"><xsl:text> is </xsl:text></xsl:when>
       <xsl:when test="self::xqx:nodeBeforeOp"><xsl:text> </xsl:text>
         <xsl:value-of select="$LESSTHAN"/><xsl:value-of select="$LESSTHAN"/>
         <xsl:text> </xsl:text></xsl:when>
```

```
      <xsl:when test="self::xqx:nodeAfterOp"><xsl:text> </xsl:text>
        <xsl:value-of select="$GREATERTHAN"/><xsl:value-of select="$GREATERTHAN"/>

        <xsl:text> </xsl:text></xsl:when>
    </xsl:choose>
    <xsl:apply-templates select="xqx:secondOperand"/>
    <xsl:value-of select="$RPAREN"/>
  </xsl:template>


  <xsl:template match="xqx:andOp | xqx:orOp">
    <xsl:value-of select="$LPAREN"/>
    <xsl:apply-templates select="xqx:firstOperand"/>
    <xsl:choose>
      <xsl:when test="self::xqx:andOp"><xsl:text> and </xsl:text></xsl:when>
      <xsl:when test="self::xqx:orOp"><xsl:text> or </xsl:text></xsl:when>
    </xsl:choose>
    <xsl:apply-templates select="xqx:secondOperand"/>
    <xsl:value-of select="$RPAREN"/>
  </xsl:template>


  <xsl:template match="xqx:unionOp | xqx:intersectOp | xqx:exceptOp">
    <xsl:value-of select="$LPAREN"/>
    <xsl:apply-templates select="xqx:firstOperand"/>
    <xsl:choose>
      <xsl:when test="self::xqx:unionOp"><xsl:text> union </xsl:text></xsl:when>
      <xsl:when test="self::xqx:intersectOp"><xsl:text> intersect
</xsl:text></xsl:when>
      <xsl:when test="self::xqx:exceptOp"><xsl:text> except </xsl:text></xsl:when>

    </xsl:choose>
    <xsl:apply-templates select="xqx:secondOperand"/>
    <xsl:value-of select="$RPAREN"/>
  </xsl:template>


  <xsl:template match="xqx:sequenceExpr">
    <xsl:for-each select=".">
      <xsl:call-template name="parenthesizedList">
        <xsl:with-param name="delimiter" select="$COMMA_NEWLINE"/>
      </xsl:call-template>
    </xsl:for-each>
  </xsl:template>
```

```
  <xsl:template match="xqx:rangeSequenceExpr">
    <xsl:value-of select="$LPAREN"/>
    <xsl:apply-templates select="xqx:startExpr"/>
    <xsl:value-of select="$TO"/>
    <xsl:apply-templates select="xqx:endExpr"/>
    <xsl:value-of select="$RPAREN"/>
  </xsl:template>


  <xsl:template match="xqx:forClause">
    <xsl:text> for </xsl:text>
    <xsl:call-template name="commaSeparatedList"/>
    <xsl:value-of select="$NEWLINE"/>
  </xsl:template>

  <xsl:template match="xqx:forClauseItem">
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="xqx:forExpr">
    <xsl:value-of select="$NEWLINE"/>
    <xsl:text>    in </xsl:text>
<xsl:apply-templates/>
  </xsl:template>


  <xsl:template match="xqx:letClause">
    <xsl:text> let </xsl:text>
    <xsl:call-template name="commaSeparatedList"/>
    <xsl:value-of select="$NEWLINE"/>
  </xsl:template>

  <xsl:template match="xqx:letClauseItem">
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="xqx:letExpr">
    <xsl:value-of select="$ASSIGN"/>
    <xsl:apply-templates/>
  </xsl:template>


  <xsl:template match="xqx:returnClause">
    <xsl:text> return </xsl:text>
```

```
  <xsl:apply-templates select="*"/>
  <xsl:value-of select="$NEWLINE"/>
</xsl:template>


<xsl:template match="xqx:whereClause">
  <xsl:text> where </xsl:text>
  <xsl:apply-templates select="*"/>
  <xsl:value-of select="$NEWLINE"/>
</xsl:template>


<xsl:template match="xqx:collation">
  <xsl:text> collation </xsl:text>
  <xsl:call-template name="quote">
    <xsl:with-param name="item">
      <xsl:value-of select="."/>
    </xsl:with-param>
  </xsl:call-template>
</xsl:template>


<xsl:template match="xqx:emptyOrderingMode">
  <xsl:value-of select="$SPACE"/>
  <xsl:value-of select="."/>
</xsl:template>


<xsl:template match="xqx:orderingKind">
  <xsl:value-of select="$SPACE"/>
  <xsl:value-of select="."/>
</xsl:template>


<xsl:template match="xqx:orderModifier">
  <xsl:apply-templates select="*"/>
</xsl:template>


<xsl:template match="xqx:orderBySpec">
  <xsl:apply-templates select="xqx:orderByExpr"/>
  <xsl:value-of select="$SPACE"/>
  <xsl:apply-templates select="xqx:orderModifier"/>
</xsl:template>
```

```
  <xsl:template match="xqx:orderByClause">
    <xsl:if test="xqx:stable">
      <xsl:text> stable</xsl:text>
    </xsl:if>
    <xsl:text> order by </xsl:text>
    <xsl:apply-templates select="xqx:orderBySpec[1]"/>
    <xsl:for-each select="xqx:orderBySpec[position() > 1]">
      <xsl:value-of select="$COMMA_SPACE"/>
      <xsl:apply-templates select="."/>
    </xsl:for-each>
    <xsl:value-of select="$NEWLINE"/>
  </xsl:template>


<!-- Surrounding FLWOR expressions with parentheses completes the set -->
  <xsl:template match="xqx:flworExpr">
    <xsl:value-of select="$NEWLINE"/>
    <xsl:value-of select="$LPAREN"/>
    <xsl:apply-templates select="*"/>
    <xsl:value-of select="$RPAREN"/>
  </xsl:template>


  <xsl:template match="xqx:ifThenElseExpr">
    <xsl:value-of select="$LPAREN"/>
    <xsl:text> if </xsl:text>
    <xsl:value-of select="$LPAREN"/>
    <xsl:apply-templates select="xqx:ifClause"/>
    <xsl:value-of select="$RPAREN"/>
    <xsl:text> then </xsl:text>
    <xsl:apply-templates select="xqx:thenClause"/>
    <xsl:text> else </xsl:text>
    <xsl:apply-templates select="xqx:elseClause"/>
    <xsl:value-of select="$RPAREN"/>
  </xsl:template>


  <xsl:template match="xqx:positionalVariableBinding">
    <xsl:text> at </xsl:text>
    <xsl:value-of select="$DOLLAR"/>
    <xsl:if test="@xqx:prefix">
      <xsl:value-of select="@xqx:prefix"/>
      <xsl:value-of select="$COLON"/>
    </xsl:if>
```

```
  <xsl:value-of select="."/>
</xsl:template>


<xsl:template match="xqx:variableBinding">
  <xsl:value-of select="$DOLLAR"/>
  <xsl:if test="@xqx:prefix">
    <xsl:value-of select="@xqx:prefix"/>
    <xsl:value-of select="$COLON"/>
  </xsl:if>
  <xsl:value-of select="."/>
  <xsl:if test="parent::xqx:typeswitchExprCaseClause">
    <xsl:text> as </xsl:text>
  </xsl:if>
</xsl:template>


<xsl:template match="xqx:typedVariableBinding" name="typedVariableBinding">
  <xsl:value-of select="$DOLLAR"/>
  <xsl:apply-templates select="xqx:varName"/>
  <xsl:apply-templates select="xqx:typeDeclaration"/>
</xsl:template>


<xsl:template match="xqx:quantifiedExprInClause">
  <xsl:apply-templates select="xqx:typedVariableBinding"/>
  <xsl:text> in </xsl:text>
  <xsl:apply-templates select="xqx:sourceExpr"/>
</xsl:template>


<xsl:template match="xqx:quantifiedExpr">
  <xsl:value-of select="$LPAREN"/>
  <xsl:value-of select="xqx:quantifier"/>
  <xsl:value-of select="$SPACE"/>
  <xsl:apply-templates select="xqx:quantifiedExprInClause[1]"/>
  <xsl:for-each select="xqx:quantifiedExprInClause[position() > 1]">
    <xsl:value-of select="$COMMA_SPACE"/>
    <xsl:apply-templates select="."/>
  </xsl:for-each>
  <xsl:text> satisfies </xsl:text>
  <xsl:apply-templates select="xqx:predicateExpr"/>
  <xsl:value-of select="$RPAREN"/>
</xsl:template>
```

```
<xsl:template match="xqx:instanceOfExpr">
  <xsl:value-of select="$LPAREN"/>
  <xsl:apply-templates select="xqx:argExpr"/>
  <xsl:text> instance of </xsl:text>
  <xsl:apply-templates select="xqx:sequenceType"/>
  <xsl:value-of select="$RPAREN"/>
</xsl:template>


<xsl:template match="xqx:castExpr">
  <xsl:value-of select="$LPAREN"/>
  <xsl:apply-templates select="xqx:argExpr"/>
  <xsl:text> cast as </xsl:text>
  <xsl:apply-templates select="xqx:singleType"/>
  <xsl:value-of select="$RPAREN"/>
</xsl:template>


<xsl:template match="xqx:castableExpr">
  <xsl:value-of select="$LPAREN"/>
  <xsl:apply-templates select="xqx:argExpr"/>
  <xsl:text> castable as </xsl:text>
  <xsl:apply-templates select="xqx:singleType"/>
  <xsl:value-of select="$RPAREN"/>
</xsl:template>


<xsl:template match="xqx:treatExpr">
  <xsl:value-of select="$LPAREN"/>
  <xsl:apply-templates select="xqx:argExpr"/>
  <xsl:text> treat as </xsl:text>
  <xsl:apply-templates select="xqx:sequenceType"/>
  <xsl:value-of select="$RPAREN"/>
</xsl:template>


<xsl:template match="xqx:typeswitchExprCaseClause">
  <xsl:text> case </xsl:text>
  <xsl:apply-templates select="xqx:variableBinding"/>
  <xsl:apply-templates select="xqx:sequenceType"/>
  <xsl:text> return </xsl:text>
  <xsl:apply-templates select="xqx:resultExpr"/>
</xsl:template>
```

```
<xsl:template match="xqx:typeswitchExprDefaultClause">
  <xsl:text> default </xsl:text>
  <xsl:apply-templates select="xqx:variableBinding"/>
  <xsl:text> return </xsl:text>
  <xsl:apply-templates select="xqx:resultExpr"/>
</xsl:template>


<xsl:template match="xqx:typeswitchExpr">
  <xsl:value-of select="$LPAREN"/>
  <xsl:text>typeswitch</xsl:text>
  <xsl:value-of select="$LPAREN"/>
  <xsl:apply-templates select="xqx:argExpr"/>
  <xsl:value-of select="$RPAREN"/>
  <xsl:apply-templates select="xqx:typeswitchExprCaseClause"/>
  <xsl:apply-templates select="xqx:typeswitchExprDefaultClause"/>
  <xsl:value-of select="$RPAREN"/>
</xsl:template>


<xsl:template match="xqx:validateExpr">
  <xsl:value-of select="$LPAREN"/>
  <xsl:text> validate </xsl:text>
  <xsl:value-of select="xqx:validationMode"/>
  <xsl:value-of select="$LBRACE"/>
  <xsl:apply-templates select="xqx:argExpr"/>
  <xsl:value-of select="$RBRACE"/>
  <xsl:value-of select="$RPAREN"/>
</xsl:template>


<xsl:template match="xqx:xpathAxis">
  <xsl:value-of select="."/>
  <xsl:value-of select="$DOUBLE_COLON"/>
</xsl:template>


<xsl:template match="xqx:predicates">
  <xsl:for-each select="*">
    <xsl:value-of select="$LBRACKET"/>
    <xsl:apply-templates select="."/>
    <xsl:value-of select="$RBRACKET"/>
  </xsl:for-each>
</xsl:template>
```

```
<xsl:template match="xqx:star">
  <xsl:value-of select="$STAR"/>
</xsl:template>


<xsl:template match="xqx:Wildcard[*]">
  <xsl:apply-templates select="*[1]"/>
  <xsl:if test="*[2]">
    <xsl:value-of select="$COLON"/>
    <xsl:apply-templates select="*[2]"/>
  </xsl:if>
</xsl:template>


<xsl:template match="xqx:Wildcard[not(*)]">
  <xsl:value-of select="$STAR"/>
</xsl:template>


<xsl:template name="simpleWildcard" match="xqx:simpleWildcard">
  <xsl:apply-templates select="xqx:star"/>
  <xsl:apply-templates select="xqx:QName"/>
</xsl:template>


<xsl:template match="xqx:textTest">
  <xsl:text>text()</xsl:text>
</xsl:template>


<xsl:template match="xqx:commentTest">
  <xsl:text>comment()</xsl:text>
</xsl:template>


<xsl:template match="xqx:anyKindTest">
  <xsl:text>node()</xsl:text>
</xsl:template>


<xsl:template match="xqx:piTest">
  <xsl:text>processing-instruction</xsl:text>
  <xsl:value-of select="$LPAREN"/>
```

```
  <xsl:value-of select="*"/>
  <xsl:value-of select="$RPAREN"/>
</xsl:template>


<xsl:template match="xqx:documentTest">
  <xsl:text>document-node</xsl:text>
  <xsl:value-of select="$LPAREN"/>
  <xsl:apply-templates select="*"/>
  <xsl:value-of select="$RPAREN"/>
</xsl:template>


<xsl:template match="xqx:nameTest">
  <xsl:if test="@xqx:prefix">
    <xsl:value-of select="@xqx:prefix"/>
    <xsl:value-of select="$COLON"/>
  </xsl:if>
  <xsl:value-of select="."/>
</xsl:template>


<xsl:template match="xqx:attributeTest">
  <xsl:text>attribute</xsl:text>
  <xsl:value-of select="$LPAREN"/>
  <xsl:for-each select="xqx:attributeName">
    <xsl:call-template name="simpleWildcard"/>
  </xsl:for-each>
  <xsl:if test="xqx:typeName">
    <xsl:value-of select="$COMMA"/>
    <xsl:apply-templates select="xqx:typeName"/>
  </xsl:if>
  <xsl:value-of select="$RPAREN"/>
</xsl:template>


<xsl:template match="xqx:elementTest">
  <xsl:text>element</xsl:text>
  <xsl:value-of select="$LPAREN"/>
  <xsl:for-each select="xqx:elementName">
    <xsl:call-template name="simpleWildcard"/>
  </xsl:for-each>
  <xsl:if test="xqx:typeName">
    <xsl:value-of select="$COMMA"/>
    <xsl:apply-templates select="xqx:typeName"/>
```

```
    </xsl:if>
    <xsl:if test="xqx:nillable">
      <xsl:value-of select="$QUESTIONMARK"/>
    </xsl:if>
    <xsl:value-of select="$RPAREN"/>
  </xsl:template>


  <xsl:template match="xqx:schemaElementTest">
    <xsl:text>schema-element</xsl:text>
    <xsl:value-of select="$LPAREN"/>
    <xsl:if test="@xqx:prefix">
      <xsl:value-of select="@xqx:prefix"/>
      <xsl:value-of select="$COLON"/>
    </xsl:if>
    <xsl:value-of select="."/>
    <xsl:value-of select="$RPAREN"/>
  </xsl:template>


  <xsl:template match="xqx:schemaAttributeTest">
    <xsl:text>schema-attribute</xsl:text>
    <xsl:value-of select="$LPAREN"/>
    <xsl:if test="@xqx:prefix">
      <xsl:value-of select="@xqx:prefix"/>
      <xsl:value-of select="$COLON"/>
    </xsl:if>
    <xsl:value-of select="."/>
    <xsl:value-of select="$RPAREN"/>
  </xsl:template>


  <!-- rewrote test expression per Bugzilla Bug #2523 -->
  <xsl:template match="xqx:stepExpr">
    <xsl:if test="preceding-sibling::xqx:stepExpr">
      <xsl:value-of select="$SLASH"/>
    </xsl:if>
    <xsl:apply-templates select="*"/>
  </xsl:template>


  <xsl:template match="xqx:parenthesizedExpr">
    <xsl:value-of select="$LPAREN"/><xsl:apply-templates/><xsl:value-of
select="$RPAREN"/>
  </xsl:template>
```

```
  <xsl:template match="xqx:filterExpr">
    <xsl:apply-templates/>
  </xsl:template>


  <!-- rewrote pathExpr template per Bugzilla Bug #2523 -->
  <xsl:template match="xqx:pathExpr">
    <xsl:apply-templates select="xqx:rootExpr | xqx:stepExpr"/>
  </xsl:template>


  <!-- To resolve Bugzilla bug #3446, we now escape NL (#xA) and TAB (#x9)
       characters in attribute values -->
  <xsl:template match="xqx:attributeConstructor">
    <xsl:value-of select="$SPACE"/>
    <xsl:apply-templates select="xqx:attributeName"/>
    <xsl:value-of select="$EQUAL"/>
    <xsl:choose>
      <xsl:when test="xqx:attributeValue">
        <xsl:call-template name="globalReplace">
          <xsl:with-param name="stringToBeFixed">
            <xsl:call-template name="globalReplace">
              <xsl:with-param name="stringToBeFixed">
                <xsl:call-template name="quote">
                  <xsl:with-param name="item">
                    <xsl:call-template name="globalReplace">
                      <xsl:with-param name="stringToBeFixed">
                        <xsl:call-template name="globalReplace">
                          <xsl:with-param name="stringToBeFixed">
                            <xsl:value-of select="xqx:attributeValue"/>
                          </xsl:with-param>
                          <xsl:with-param
name="toBeReplaced"><xsl:text>{</xsl:text></xsl:with-param>
                          <xsl:with-param
name="replacement"><xsl:text>{{</xsl:text></xsl:with-param>
                        </xsl:call-template>
                      </xsl:with-param>
                      <xsl:with-param
name="toBeReplaced"><xsl:text>}</xsl:text></xsl:with-param>
                      <xsl:with-param
name="replacement"><xsl:text>}}</xsl:text></xsl:with-param>
                    </xsl:call-template>
                  </xsl:with-param>
```

```
              </xsl:call-template>
            </xsl:with-param>
            <xsl:with-param name="toBeReplaced" select="'&#xA;'"/>
            <xsl:with-param name="replacement">&amp;#xA;</xsl:with-param>
          </xsl:call-template>
        </xsl:with-param>
        <xsl:with-param name="toBeReplaced" select="'&#x9;'"/>
        <xsl:with-param name="replacement">&amp;#x9;</xsl:with-param>
      </xsl:call-template>
    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="$DOUBLEQUOTE"/>
      <xsl:for-each select="./xqx:attributeValueExpr/xqx:*">
        <xsl:value-of select="$LBRACE"/>
          <xsl:apply-templates select="."/>
        <xsl:value-of select="$RBRACE"/>
      </xsl:for-each>
      <xsl:value-of select="$DOUBLEQUOTE"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>


<xsl:template match="xqx:namespaceDeclaration">
  <xsl:text> xmlns</xsl:text>
  <xsl:if test="xqx:prefix">
    <xsl:text>:</xsl:text>
    <xsl:value-of select="xqx:prefix"/>
  </xsl:if>
  <xsl:value-of select="$EQUAL"/>
  <xsl:call-template name="quote">
    <xsl:with-param name="item" select="xqx:uri"/>
  </xsl:call-template>
</xsl:template>

<xsl:template match="xqx:attributeList">
  <xsl:apply-templates select="*"/>
</xsl:template>


<xsl:template match="xqx:elementContent">
  <xsl:for-each select="*">
    <xsl:if test="not(self::xqx:elementConstructor)">
       <xsl:value-of select="$LBRACE"/>
    </xsl:if>
```

```
    <xsl:apply-templates select="."/>
    <xsl:if test="not(self::xqx:elementConstructor)">
       <xsl:value-of select="$RBRACE"/>
    </xsl:if>
  </xsl:for-each>
</xsl:template>


<xsl:template match="xqx:elementConstructor">
  <xsl:value-of select="$LESSTHAN"/>
  <xsl:apply-templates select="xqx:tagName"/>
  <xsl:apply-templates select="xqx:attributeList"/>
  <xsl:value-of select="$GREATERTHAN"/>
  <xsl:apply-templates select="xqx:elementContent"/>
  <xsl:value-of select="$LESSTHAN"/>
  <xsl:value-of select="$SLASH"/>
  <xsl:apply-templates select="xqx:tagName"/>
  <xsl:value-of select="$GREATERTHAN"/>
</xsl:template>


<xsl:template match="xqx:tagNameExpr">
  <xsl:value-of select="$LBRACE"/>
  <xsl:apply-templates select="*"/>
  <xsl:value-of select="$RBRACE"/>
</xsl:template>


<xsl:template match="xqx:computedElementConstructor">
  <xsl:text> element </xsl:text>
  <xsl:apply-templates select="xqx:tagName"/>
  <xsl:apply-templates select="xqx:tagNameExpr"/>
  <xsl:value-of select="$LBRACE"/>
  <xsl:apply-templates select="xqx:contentExpr"/>
  <xsl:value-of select="$RBRACE"/>
</xsl:template>


<xsl:template match="xqx:computedAttributeConstructor">
  <xsl:text> attribute </xsl:text>
  <xsl:apply-templates select="xqx:tagName"/>
  <xsl:apply-templates select="xqx:tagNameExpr"/>
  <xsl:value-of select="$LBRACE"/>
  <xsl:apply-templates select="xqx:valueExpr"/>
  <xsl:value-of select="$RBRACE"/>
```

```
  </xsl:template>


  <xsl:template match="xqx:computedDocumentConstructor">
    <xsl:text> document {</xsl:text>
    <xsl:apply-templates select="*"/>
    <xsl:text> }</xsl:text>
  </xsl:template>


  <xsl:template match="xqx:computedTextConstructor">
    <xsl:text> text</xsl:text>
    <xsl:value-of select="$LBRACE"/>
    <xsl:apply-templates select="*"/>
    <xsl:value-of select="$RBRACE"/>
  </xsl:template>


  <xsl:template match="xqx:computedCommentConstructor">
    <xsl:text> comment</xsl:text>
    <xsl:value-of select="$LBRACE"/>
    <xsl:apply-templates select="*"/>
    <xsl:value-of select="$RBRACE"/>
  </xsl:template>


  <xsl:template match="xqx:piTargetExpr">
    <xsl:value-of select="$LBRACE"/>
    <xsl:apply-templates select="*"/>
    <xsl:value-of select="$RBRACE"/>
  </xsl:template>


<!-- Move value braces into computedPIConstructor template from this template,
Bugzilla bug #3442 -->
  <xsl:template match="xqx:piValueExpr">
    <xsl:apply-templates select="*"/>
  </xsl:template>


<!-- Move value braces into this template from piValueExpr template, Bugzilla bug
#3442 -->
  <xsl:template match="xqx:computedPIConstructor">
    <xsl:text> processing-instruction </xsl:text>
    <xsl:value-of select="xqx:piTarget"/>
```

```
    <xsl:apply-templates select="xqx:piTargetExpr"/>
    <xsl:value-of select="$LBRACE"/>
    <xsl:apply-templates select="xqx:piValueExpr"/>
    <xsl:value-of select="$RBRACE"/>
  </xsl:template>


  <xsl:template match="xqx:unorderedExpr">
    <xsl:text> unordered</xsl:text>
<xsl:value-of select="$LBRACE"/>
    <xsl:text> </xsl:text>
    <xsl:apply-templates select="*"/>
    <xsl:text> </xsl:text>
    <xsl:value-of select="$RBRACE"/>
  </xsl:template>


  <xsl:template match="xqx:orderedExpr">
    <xsl:text> ordered</xsl:text>
<xsl:value-of select="$LBRACE"/>
    <xsl:text> </xsl:text>
    <xsl:apply-templates select="*"/>
    <xsl:text> </xsl:text>
    <xsl:value-of select="$RBRACE"/>
  </xsl:template>


  <xsl:template match="xqx:versionDecl">
    <xsl:text> xquery version </xsl:text>
    <xsl:call-template name="quote">
      <xsl:with-param name="item" select="xqx:version"/>
    </xsl:call-template>
    <xsl:value-of select="$SEPARATOR"/>
    <xsl:value-of select="$NEWLINE"/>
  </xsl:template>


  <xsl:template match="xqx:namespaceDecl">
    <xsl:text>declare namespace </xsl:text>
    <xsl:value-of select="xqx:prefix"/>
    <xsl:value-of select="$EQUAL"/>
    <xsl:call-template name="quote">
      <xsl:with-param name="item" select="xqx:uri"/>
    </xsl:call-template>
  </xsl:template>
```

```
<xsl:template match="xqx:defaultNamespaceDecl">
  <xsl:text>declare default </xsl:text>
  <xsl:value-of select="xqx:defaultNamespaceCategory"/>
  <xsl:text> namespace </xsl:text>
  <xsl:call-template name="quote">
    <xsl:with-param name="item" select="xqx:uri"/>
  </xsl:call-template>
</xsl:template>


<xsl:template match="xqx:defaultCollationDecl">
  <xsl:text>declare default collation </xsl:text>
  <xsl:call-template name="quote">
    <xsl:with-param name="item" select="."/>
  </xsl:call-template>
</xsl:template>


<xsl:template match="xqx:baseUriDecl">
  <xsl:text>declare base-uri </xsl:text>
  <xsl:call-template name="quote">
    <xsl:with-param name="item" select="."/>
  </xsl:call-template>
</xsl:template>

<xsl:template match="xqx:constructionDecl">
  <xsl:text>declare construction </xsl:text>
  <xsl:value-of select="."/>
</xsl:template>


<xsl:template match="xqx:orderingModeDecl">
  <xsl:text>declare ordering </xsl:text>
  <xsl:value-of select="."/>
</xsl:template>


<xsl:template match="xqx:emptyOrderingDecl">
  <xsl:text>declare default order </xsl:text>
  <xsl:value-of select="."/>
</xsl:template>
```

```
<xsl:template match="xqx:copyNamespacesDecl">
  <xsl:text>declare copy-namespaces </xsl:text>
  <xsl:value-of select="xqx:preserveMode"/>
  <xsl:value-of select="$COMMA"/>
  <xsl:value-of select="xqx:inheritMode"/>
</xsl:template>


<xsl:template match="xqx:optionDecl">
  <xsl:text>declare option </xsl:text>
  <xsl:apply-templates select="xqx:optionName"/>
  <xsl:value-of select="$SPACE"/>
  <xsl:call-template name="quote">
    <xsl:with-param name="item" select="xqx:optionContents"/>
  </xsl:call-template>
</xsl:template>


<xsl:template match="xqx:voidSequenceType">
  <xsl:text>empty-sequence()</xsl:text>
</xsl:template>


<xsl:template match="xqx:occurrenceIndicator">
  <xsl:value-of select="."/>
</xsl:template>


<xsl:template match="xqx:anyItemType">
  <xsl:text>item()</xsl:text>
</xsl:template>


<xsl:template match="xqx:sequenceType">
  <xsl:apply-templates select="*"/>
</xsl:template>


<xsl:template match="xqx:singleType">
  <xsl:apply-templates select="xqx:atomicType"/>
  <xsl:if test="xqx:optional">
    <xsl:text>?</xsl:text>
  </xsl:if>
</xsl:template>
```

```
  <xsl:template match="xqx:typeDeclaration">
    <xsl:text> as </xsl:text>
    <xsl:apply-templates select="*"/>
  </xsl:template>


  <xsl:template match="xqx:varDecl">
    <xsl:text>declare variable </xsl:text>
    <xsl:value-of select="$DOLLAR"/>
    <xsl:apply-templates select="xqx:varName"/>
    <xsl:apply-templates select="xqx:typeDeclaration"/>
    <xsl:if test="xqx:external">
      <xsl:text> external </xsl:text>
    </xsl:if>
    <xsl:if test="xqx:varValue">
      <xsl:value-of select="$ASSIGN"/>
      <xsl:apply-templates select="xqx:varValue"/>
    </xsl:if>
  </xsl:template>


<!-- Part of fix for Bugzilla bug #3520 -->
  <xsl:template match="xqx:targetLocation">
    <xsl:choose>
      <xsl:when test="position()=1"> at </xsl:when>
      <xsl:otherwise>,&#xD;  </xsl:otherwise>
    </xsl:choose>
    <xsl:call-template name="quote">
      <xsl:with-param name="item" select="."/>
    </xsl:call-template>
  </xsl:template>


<!-- Modified to fix Bugzilla bug #3520 -->
  <xsl:template match="xqx:schemaImport">
    <xsl:text> import schema </xsl:text>
    <xsl:if test="xqx:defaultElementNamespace">
      <xsl:text> default element namespace </xsl:text>
    </xsl:if>
    <xsl:if test="xqx:namespacePrefix">
      <xsl:text> namespace </xsl:text>
      <xsl:value-of select="xqx:namespacePrefix"/>
      <xsl:value-of select="$EQUAL"/>
    </xsl:if>
```

```
    <xsl:call-template name="quote">
      <xsl:with-param name="item" select="xqx:targetNamespace"/>
    </xsl:call-template>
    <xsl:apply-templates select="xqx:targetLocation"/>
  </xsl:template>


<!-- Modified to fix Bugzilla bug #3520 -->
  <xsl:template match="xqx:moduleImport">
    <xsl:text> import module </xsl:text>
    <xsl:if test="xqx:namespacePrefix">
      <xsl:text> namespace </xsl:text>
      <xsl:value-of select="xqx:namespacePrefix"/>
      <xsl:value-of select="$EQUAL"/>
    </xsl:if>
    <xsl:call-template name="quote">
      <xsl:with-param name="item" select="xqx:targetNamespace"/>
    </xsl:call-template>
    <xsl:apply-templates select="xqx:targetLocation"/>
  </xsl:template>


  <xsl:template match="xqx:param">
    <xsl:value-of select="$DOLLAR"/>
    <xsl:apply-templates select="xqx:varName"/>
    <xsl:apply-templates select="xqx:typeDeclaration"/>
  </xsl:template>


  <xsl:template match="xqx:paramList">
    <xsl:call-template name="parenthesizedList"/>
  </xsl:template>


  <xsl:template match="xqx:functionBody">
    <xsl:value-of select="$NEWLINE"/>
    <xsl:value-of select="$LBRACE"/>
    <xsl:value-of select="$NEWLINE"/>
    <xsl:apply-templates/>
    <xsl:value-of select="$NEWLINE"/>
    <xsl:value-of select="$RBRACE"/>
  </xsl:template>


  <xsl:template match="xqx:functionDecl">
```

```
    <xsl:text>declare function </xsl:text>
    <xsl:apply-templates select="xqx:functionName"/>
    <xsl:apply-templates select="xqx:paramList"/>
    <xsl:apply-templates select="xqx:typeDeclaration"/>
    <xsl:apply-templates select="xqx:functionBody"/>
    <xsl:if test="xqx:externalDefinition">
      <xsl:text> external </xsl:text>
    </xsl:if>
  </xsl:template>


  <xsl:template match="xqx:queryBody">
    <xsl:apply-templates select="*"/>
    <xsl:value-of select="$NEWLINE"/>
  </xsl:template>

  <xsl:template match="xqx:moduleDecl">
    <xsl:text> module namespace </xsl:text>
    <xsl:value-of select="xqx:prefix"/>
    <xsl:value-of select="$EQUAL"/>
    <xsl:call-template name="quote">
      <xsl:with-param name="item" select="xqx:uri" />
    </xsl:call-template>
    <xsl:value-of select="$SEPARATOR"/>
    <xsl:value-of select="$NEWLINE"/>
  </xsl:template>


  <xsl:template match="xqx:prolog">
    <xsl:for-each select="*">
      <xsl:apply-templates select="."/>
      <xsl:value-of select="$SEPARATOR"/>
      <xsl:value-of select="$NEWLINE"/>
    </xsl:for-each>
  </xsl:template>


  <xsl:template match="xqx:libraryModule">
    <xsl:apply-templates select="xqx:moduleDecl"/>
    <xsl:apply-templates select="xqx:prolog"/>
  </xsl:template>


  <xsl:template match="xqx:mainModule">
    <xsl:apply-templates select="xqx:prolog"/>
```

```
    <xsl:apply-templates select="xqx:queryBody"/>
  </xsl:template>


  <xsl:template match="xqx:module" priority="2">
    <xsl:apply-templates select="*"/>
  </xsl:template>


  <xsl:template match="/xqx:*">
   <xsl:message terminate="yes">Incorrect XQueryX: The only top-level element
permitted is xqx:module</xsl:message>
  </xsl:template>


</xsl:stylesheet>
```

Appendix C. The `application/xquery+xml` Media Type

This Appendix specifies the media type for XQueryX Version 1.0. XQueryX is the XML syntax of a language, XQuery, for querying over data from XML data sources, as specified in [XQuery 1.0: An XML Query Language].

Specification of media types is described in [RFC 3023].

C.1. Introduction

This document, together with its normative references, defines the XML syntax for the XML Query language XQuery Version 1.0. This Appendix specifies the `application/xquery+xml` media type, which is intended to be used for transmitting queries expressed in the XQueryX syntax.

This media type is being submitted to the IESG for review, approval, and registration with IANA.

This document was prepared by members of the W3C XML Query Working Group. Please send comments to public-qt-comments@w3.org, a public mailing list with archives at http://lists.w3.org/Archives/Public/public-qt-comments.

C.2. Registration of MIME Media Type `application/xquery+xml`

MIME media type name: `application`

MIME subtype name: `xquery+xml`

Required parameters: none

Optional parameters: `charset`

This parameter has identical semantics to the `charset` parameter of the `application/xml` media type as specified in [RFC 3023].

C.2.1. Encoding Considerations

By virtue of XSLT content being XML, it has the same considerations when sent as "`application/xquery+xml`" as does XML. See [RFC 3023], section 3.2.

C.2.2. Security Considerations

Queries written in XQuery may cause arbitrary URIs or IRIs to be dereferenced. Therefore, the security issues of [RFC3987] Section 8 should be considered. In addition, the contents of resources identified by `file:` URIs can in some cases be accessed, processed and returned as results. XQuery expressions can invoke any of the functions defined in XQuery 1.0 and XPath 2.0 Functions and Operators, including `file-exists()`; a `doc()` function also allows local filesystem probes as well as access to any URI-defined resource accessible from the system evaluating the XQuery expression.

XQuery is a full declarative programming language, and supports user-defined functions, external function libraries (modules) referenced by URI, and system-specific "native" functions.

Arbitrary recursion is possible, as is arbitrarily large memory usage, and implementations may place limits on CPU and memory usage, as well as restricting access to system-defined functions.

The XML Query Working group is working on a facility to allow XQuery expressions to be used to create and update persistent data. Untrusted queries should not be given write access to data.

Furthermore, because the XQuery language permits extensions, it is possible that application/xquery may describe content that has security implications beyond those described here.

C.2.3. Interoperability Considerations

See .

C.2.4. Published specification

This media type registration is for XQueryX documents as described by the XQueryX 1.0 specification, which is located at http://www.w3.org/TR/xqueryx/. It is also appropriate to use this media type with later versions of the XQueryX language.

C.2.5. Applications That Use This Media Type

The public XQuery Web page lists more than two dozen implementations of the XQuery language, both proprietary and open source. Some of these are known to support XQueryX.

This new media type is being registered to allow for deployment of XQueryX on the World Wide Web.

There is no experimental, vendor specific, or personal tree predecessor to "application/xquery+xml", reflecting the fact that no applications currently recognize it. This new type is being registered in order to allow for the expected deployment of XQueryX 1.0 on the World Wide Web, as a first class XML application.

C.2.6. Additional Information

C.2.6.1. Recognizing XQuery Files ("Magic Numbers")

Although no byte sequences can be counted on to consistently identify XQueryX, XQueryX documents will have the sequence "http://www.w3.org/2005/XQueryX" to identify the XQueryX namespace. This sequence will normally be found in a namespace attribute of the first element in the document.

C.2.6.2. File Extensions

The most common file extension in use for XQueryX is `.xqx`.

C.2.6.3. Macintosh File Type Code(s)

The appropriate Macintosh file type code is `TEXT`.

C.2.7. Person and Email Address to Contact For Further Information

Jim Melton, Oracle Corp., jim.melton@oracle.com

C.2.8. Intended Usage

COMMON

C.2.9. Restrictions on usage

The intended usage of this media type is for interchange of XQueryX expressions.

C.2.10. Author/Change Controller

XQuery was produced by, and is maintained by, the World Wide Web Consortium's XML Query Working Group. The W3C has change control over this specification.

C.2.11. Fragment Identifiers

For documents labeled as "`application/xquery+xml`", the fragment identifier notation is exactly that for "`application/xml`", as specified in [RFC 3023].

Appendix D. Change log (Non-Normative)

This appendix lists the changes that have been made to this specification since the publication of the Proposed Recommendation Draft on 21 November 2006.

- The MIME appendix was updated to normalize the text regarding security considerations and to create a reference to the XQueryX spec that will be visible when the MIME type is registered with IETF.
- A few very minor typographical errors were corrected and one or two sentences were reworded to be consistent with other sentences having similar purposes.

This page is intentionally left blank.