XQuery 1.0 is Nearing Completion

Andrew Eisenberg
IBM, Westford, MA 01886
andrew.eisenberg@us.ibm.com

Jim Melton
Oracle Corp., Sandy, UT 84093
jim.melton@acm.org

Introduction

XQuery is a query language designed for querying real and virtual XML documents and collections of these documents. Its development began in the second half of 1999. We provided an early look at XQuery in Dec. 2002 [1]. XQuery 1.0 is now approaching its publication as a W3C Recommendation, and we would like to update you on its progress. We can speak to this area with even more authority than we did last time, as we both became co-chairs of the W3C XML Query Working Group [2] in summer 2004.

Paul Cotton (Microsoft), who chaired the group since its inception, stepped down from this role in October. His role in other consortia didn't allow him to stay with XQuery 1.0 all the way though its publication as a Recommendation, although he certainly wanted to do so. Paul deserves a great deal of credit for the leading role that he has played in the development of XQuery.

In his article, we concentrate on the changes that have taken place to XQuery since our earlier article. If you are unfamiliar with XQuery, then you may want to take a look at our earlier article before proceeding.

XQuery Status

The following documents [3] became Candidate Recommendations (CR) in November 2005.

- XQuery 1.0: An XML Query Language
- XML Path Language (XPath) 2.0
- XSL Transformations (XSLT) Version 2.0
- XQuery 1.0 and XPath 2.0 Data Model (XDM)
- XQuery 1.0 and XPath 2.0 Functions and Operators
- XQuery 1.0 and XPath 2.0 Formal Semantics
- XSLT 2.0 and XQuery 1.0 Serialization
- XML Syntax for XQuery 1.0 (XQueryX)

The XML Query WG has worked closely with the XSL WG on most of these documents.

Most of the documents underwent two Last Call Working Draft (WD) reviews, and a couple of them underwent three such reviews. In the last review, the WGs responded to approximately 600 comments.

The purpose of CR is to gain implementation experience and give a WG confidence that its specification is complete and unambiguous. To this end, the XML Query WG began the development of a test suite in the summer of 2004. The XML Query Test Suite [4] now covers about 75% of the features that make up XQuery. It will take several months for this test suite to be completed and to get reports back from implementers.

With luck, and some hard work on the part of a number of people, XQuery will become a W3C Recommendation before the end of 2006.

The XQuery 1.0 and XPath 2.0 Data Model (XDM)

XDM defines five types beyond those defined in XML Schema Part 2 [5]. Two of them were discussed in our previous article: `xdt:dayTimeDuration` and `xdt:yearMonthDuration`, where `xdt` is a prefix for the namespace `http://www.w3.org/2005/xpath-datatypes`.

`xdt:untyped` is assigned to element nodes that have not been validated or have been validated in skip mode. `xdt:untyped` is also the type assigned to a constructed element when the construction mode is `strip` (discussed later). All of the children of an element that is annotated as `xdt:untyped` are annotated as `xdt:untyped` as well.

`xdt:untypedAtomic` is assigned to values that are atomic, but which do not have a more specific type. An attribute that has been validated in skip mode are assigned this type.

`xdt:anyAtomicType` has `xs:anySimpleType` as its base type, and is the type from which all primitive atomic types are derived. These include types such as `xs:string`, `xs:float`, and `xdt:untypedAtomic`. This type is abstract in nature, as no values will be annotated with this type. From XQuery's point of view, this type has been inserted into the XML Schema type hierarchy.

Serialization

Since we wrote our earlier article, the WGs have created a new document, XSLT 2.0 and XQuery 1.0 Serialization [6]. The material in this document was removed from XSLT 2.0, so that it could be shared by XQuery 1.0. This document defines the XML, XHTML, HTML, and TEXT output methods. XQuery 1.0 makes use of only the XML output method, while XSLT uses all of them.

A value of the XQuery 1.0 and XPath 2.0 Data Model (XDM) may be provided for serialization. Sequence Normalization is performed, followed by markup generation, character expansion, and encoding.

Sequence Normalization is defined in several steps, transforming a data model instance—a sequence of values and nodes—into a single document node. Atomic values are cast into strings and then into text nodes. Document nodes are discarded, but their children are retained. It is a serialization error if an attribute node that is not a child of an element node is placed into the resulting document.

Serialization defines a number of parameters that influence the result that is produced. `omit-xml-declaration`, for example, can have either `yes` or `no` for its value. Not all parameters are used by a given output method.

The XML output method generates a well-form XML document entity if the result of sequence normalization is a document node with a single element node child and no text node children. Otherwise, a well-formed XML external general parsed entity is generated. The specification doesn't say how to form these entities. Instead, it requires that the same data model instance be produced by parsing the result and using the resulting infoset to generate a data model instance. Well, not exactly the same: it describes ways in which they are allowed to differ, such as the order in which attribute nodes appear.

No attempt is made to preserve the type annotations during serialization. If the result is XML Schema validated, then new type annotations will be created.

XQuery

XQuery 1.0 is almost a proper superset of XPath 2.0—XQuery 1.0 does not use XPath's namespace nodes and does not support XPath's namespace axis.

Inputs to XQuery Processing

The data model instances that XQuery can operate on can be provided in a number of ways. Our earlier article described the context item, denoted by ".", and the `fn:doc` and `fn:collection` functions. The `xf:input` function that we described earlier has been dropped in favor of external variables.

A variant of the `fn:collection` function without an argument has been introduced to refer to a default collection that may be supplied by the host environment.

Variables may be provided by an implementation for use in a query. A query may also define external variables and expect values for these variables to be provided by the host environment. The variable declaration may include a type for the variable. If it does not, then the host environment provides the variable's type as well as its value.

The following query might be executed with the `$custName` variable bound to "Big Box".

```
declare variable $custName as xs:string
external;

fn:doc('orders.xml')
   /orders/order[@cust=$custName]
```

$\rightarrow$

```
<order id='444378' cust='Big Box'>
   ...
</order>
```

Steps that Return Atomic Values

In XPath 1.0, the result of a step in a path expression was a sequence of nodes in document order with duplicates removed. XQuery 1.0 and XPath 2.0 allow the final step in a path expression to produce a sequence of atomic values. A query for the cities and states of all California employees can be written as:

```
//employee[address/state='CA']
  /address/concat(city, ', ', state)
```

rather than:

```
for $a in //employee[address/state="CA"]
            /address
return concat($a/city, ', ', $a/state)
```

Declarations in the XQuery Prolog

A number of declarations have been added to XQuery's prolog. Some of these are the boundary-space declaration, base URI declaration, construction declaration, copy namespaces declaration, and option declaration. We'll discuss a couple of these in this section.

Boundary Space

The `boundary-space` declaration has values of `preserve` and `strip`, and determines whether boundary whitespace is preserved by element constructors. Let's look at an example:

```
declare boundary-space preserve;

<test>   <inner-element/>   </test>
```

The test element that is returned has 3 children; a text node containing several spaces, an element node, and another text node. If `strip` had been chosen, then the element node would be the only child. If this declaration is not used, then `strip` is the default.

Construction

The `construction` declaration also has values of `preserve` and `strip`. Here, a user chooses whether type annotations are preserved in the construction of new element and document nodes. If `strip` is chosen, then the constructed element node and all of its children are annotated with `xdt:untyped`, and all of its attribute nodes are annotated with `xdt:untypedAtomic`. If `preserve` is chosen, then the constructed element node is annotated with `xdt:anyType`, and all of its element nodes and attribute nodes retain their existing annotations.

Option

An `option` declaration is one of several extension mechanisms that XQuery provides to implementers. An option declaration contains a QName and string. If the QName is recognized by an implementation, then it can have whatever effect on the processing of the query the implementer chooses. If it is not recognized, then it is ignored. In this way, the extensions of one implementation will not cause execution on another implementation to fail.

Let's consider an extension that allows a user to set a timeout value, in seconds, after which the query will stop and return an error.

```
declare namespace myxquery='...';
declare option myxquery:timeout '10';

for $e in //employees ...
```

Expressions

`castable`, `extension`, `ordered`, and `unordered` have been added to the set of XQuery expressions and the syntax has been changed just a bit for `cast`, node comparison, and `validate`.

expression type	**expression syntax**
cast	***expr*** **cast as** ***type***
castable	***expr*** **castable as** ***type***

validate	**validate {** ***expr*** **}**
	validate lax { ***expr*** **}**
	validate strict { ***expr*** **}**

node comparison	**is** ***(isnot was dropped)***
extension	***(see below)***
ordered	**ordered {** ***expr*** **}**
unordered	**unordered {** ***expr*** **}**

Castable

`castable` returns a Boolean value that indicates whether the value provided can be successfully cast to the type provided. Without this expression, a user would not be able to prevent the failure of a cast becoming a failure of the entire query. (Exception handling is something that might be considered in a future version of XQuery.)

Ordered and Unordered

An `ordered` expression sets the ordering mode to `ordered` for the expression that it contains. An `unordered` expression sets the ordering mode to `unordered`.

Path expressions that include a "`/`" or "`//`" operator or a step, set expressions (`union`, `intersect`, and `except`), and FLWOR expressions without an `order by` clause are sensitive to the setting of the ordering mode. When it is `ordered`, each produces its sequence of items in document order. When it is `unordered`, each produces its sequence of items in an arbitrary order. Relaxing the order of the items may allow an optimizer to choose a lower-cost strategy for evaluating the query.

The initial ordering mode can be set by a user in the query prolog. If it is not set, then the default ordering mode is `ordered`.

The following query returns New York employees in an arbitrary order, but it uses ordering in the inner path expression to select employees whose last title is "VP".

```
declare ordering unordered;

for $e in ordered {
  //employee[titles/title[last()] = 'VP'] }
where $e[location/@state='NY']
return $e
```

Validate

The `validate` expression applies XML Schema validation to its argument. Its argument is first converted into an infoset, discarding any type annotations that it might have contained. The result of validation is a new element (with new contents and new identity) with type annotations. If validation is not successful, then a dynamic type error is raised.

Type annotations can be applied to a constructed element using the validate expression:

```
validate { <myco:employee id='440612'>
              <name>Augustus Child</name>
              .
              .
              .
           </myco:employee>
         }
```

In this case, the `myco` schema must contain a globally defined element `employee`. The `name` element in the constructed element has type `xdt:untyped`, while in the validated result it might have type `myco:nameType`.

Extension

An extension expression is another extension mechanism provided to implementers by XQuery. Where an `option` declaration has an effect for the entire query, an extension expression has a narrower scope. Let's use the following example to explain this construct.

```
declare namespace xq1="...";
declare namespace xq2="...";

for $e in //employee[name='Jon Postel']
return (# xq1:prose English #)
       (# xq2:roman lower-case #)
       { $e/badge cast as xs:string }
```

These *pragmas* "`(# … #)`", if they are recognized, might change the behavior of casting values to strings. This query might produce "One Hundred Fifty Four" if it recognizes `xq1:prose`, "cliv" if it recognizes `xq2:roman`, and "154" if it recognizes neither of them. The expression in curly braces "`{}`" can be omitted. If it is omitted and none of the pragmas is recognized, then an error is raised.

URI Values

XQuery has long allowed the type promotion of numeric values, from `xs:decimal` to `xs:float` and from `xs:float` to `xs:double`. Since our earlier article, XQuery has added promotion from `xs:anyURI` to `xs:string`.

Without this change, a query on an untyped document written as:

```
let $xq := 'http://www.w3.org/TR/xquery/'
return count(//bib[ref=$xq])
```

would cause a type error for a typed document due to the comparison of an `xs:anyURI` and an `xs:string` value. It would have to be rewritten as:

```
let $xq := 'http://www.w3.org/TR/xquery/'
return count(//bib[ref=xs:anyURI($xq)])
```

Types

Some of the type designators have changed since our last article. Rather than going through BNF, we'll just look a number of examples:

`xs:integer?`	a sequence of zero or one integer
`element()+`	a sequence of one or more elements
`node()*`	a sequence of zero or more nodes
`item()+`	one or more items
`attribute()`	an attribute (single) of any name and type
`element (myco:address)`	an element with name `myco:address`
`element (*, myco:addrType)`	an element of any name, with type `myco:addrType`
`schema-element(zip)`	an element named `zip` (or in a substitution group headed by `zip`) with a type annotation that matches the type of `zip` element

A type designator might be used as follows:

```
//employee
  [* instance of
     element (*, myco:addrType)

  ]
```

Earlier versions of XQuery allowed reference to be made to element and attributes that were locally declared in a schema, but this feature was dropped.

FLWOR Expression

The FLWR (for, let, where, return) expression has become the FLWOR expression (where "O" stands for "order by".

Each `order by` clause can contain multiple sort keys, each of which contains an expression and may contain an indication of whether the sorting should be stable, whether it should be ascending or descending, whether an empty sequence is considered greater than or less than any item, and whether a collation sequence other than the default collation sequence should be used.

Each expression in the `order by` clause is evaluated for each of the bindings of the variables in the `for` and `let` clauses that are not eliminated by the `where` clause. If any expression produces a sequence of more than one item, then an error is returned. Any values that are of type `xdt:untypedAtomic` are cast to `xs:string`. If, for any sort key, the values differ in type (after considering subtype substitution and type promotion), then an error is returned.

The following query returns recently hired employees ordered first by their years of education and then by their department.

```
for $e in doc('employees.xml')//employee
where current-date() - $e/hireDate
      < xdt:dayTimeDuration('P60D')
order by
   $e/HSYears + $e/CollegeYears descending,
   $e/dept empty greatest
return $e
```

The choice of whether an empty sequence is greater than or less than an item can be made in the query prolog. An XQuery implementation can choose either of these as its default behavior.

The FLWOR expression also gained an `at` clause that binds the position of the item in the sequence at the same time that it binds the value of that item.

The following query returns the 10 employees that have been with the company the longest:

```
for $e at $p in
   (for $oe in //employee
    order by $oe/@hireDate descending
    return $oe)
where $p <= 10
return $e
```

In-scope Namespaces

XQuery has chosen not to support namespace nodes and a namespace axis, as XPath 1.0 did. Instead, XQuery associates a set of in-scope namespace bindings with its nodes.

XQuery also has a set of statically known namespaces, which are used when resolving its QNames. These statically known namespaces include `fn`, `xml`, `xs`, `xsi`, `xdt`, and `local`. An implementation may add its own namespace bindings, and a user may add to all of these bindings in the query prolog:

```
declare namespace
        myco="http://www.example.com/myco";

<myco:result> { for ... } </myco:result>
```

The in-scope namespaces may affect how an element node is serialized and may also affect the behavior of a small number of functions. The node constructed in this example has one namespace binding associated with it. The namespace for `myco` is taken from the statically known namespaces when the node is constructed.

When a node is constructed, its namespace bindings include the one used in the element name, those used in the attribute names, those defined by namespace declaration attributes, and those in namespace attribute declarations of enclosing element constructors that have not been overwritten.

Let's consider the following example:

```
import schema namespace hr="...";

validate strict {
   <hr:employee>
      <hr:skill xsi:type="xs:string">
         unicycling
      </hr:skill>
   </hr:employee>
}
```

This query will raise an error, because a binding for `xs` will not appear in the infoset that is validated. The `xsi:type` attribute is given no special consideration by XQuery. "xs:string" is just an untyped attribute value, it is not seen as a QName, and so `xs` does not get added to the in-scope namespaces. This means that it does not become part of the infoset. This query can be fixed by changing the start tag as follows:

```
<hr:employee xmlns:xs
      ="http://www.w3.org/2001/XMLSchema">
   .
   .
   .
</hr:employee>
```

Finer-grained control over the in-scope namespaces of constructed nodes is available to a user via the `copy-namespaces` declaration in the query prolog.

Modules

A library module is a collection of variables and functions in a target namespace that can be imported into a query.

```
module namespace univ
       ="http://www.example.com/university";

declare function univ:gpa
   ($e as element (student)) as xs:decimal
   { for ... } ;
```

This function could be invoked in a query in the following way:

```
import module namespace univ
       ="http://www.example.com/university";
declare variable $id external;

univ:gpa(//student[id=$id])
```

XQueryX

XQueryX [9] defines an XML representation of XQuery. It defines an element structure that mirrors the abstract syntax of XQuery. The definition of XQueryX has changed quite a bit since we showed it to you last. Example 1 contains a simple XQuery and the corresponding XQueryX representation.

While XQueryX is harder for a human to read and write than XQuery, it does have several useful properties. It is easily generated by tools and layered applications, it is easily embedded within larger XML documents, and it allows "queries on queries".

Of course, all changes made to XQuery apply equally to XQueryX. But there is another fairly important change that has been made to XQueryX. When we last showed it to you, the XML Schema that defines the XQueryX syntax was based on a sort of type hierarchy that turned out to be difficult to maintain as new features were added to the language, and also somewhat difficult for human readers to keep in their minds. That hierarchical design has been replaced with one based on XML Schema's substitution groups. This sort of approach is more readily extensible when new language features are created, and also more familiar to Schema experts.

```
for $b in .//book
return $b/title

→

<xqx:module
   xmlns:xqx="http://www.w3.org/2005/XQueryX"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <xqx:mainModule>
    <xqx:queryBody>
      <xqx:flworExpr>
        <xqx:forClause>
          <xqx:forClauseItem>
            <xqx:typedVariableBinding>
              <xqx:varName>b</xqx:varName>
            </xqx:typedVariableBinding>
            <xqx:forExpr>
              <xqx:pathExpr>
                <xqx:argExpr>
                  <xqx:contextItemExpr/>
                </xqx:argExpr>
                <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:returnClause>
           .
           .
           .
        </xqx:returnClause>
      </xqx:flworExpr>
    </xqx:queryBody>
  </xqx:mainModule>
</xqx:module>
```

Example 1 – Equivalent XQuery and XQueryX

Several minor changes were also made to XQueryX's Schema. The most significant change is a new Schema element, <xqx:xquery>, used for a trivial embedding of XQuery (human-readable) text into XML documents.

XQuery and XQueryX Conformance

Both XQuery and XQueryX have conformance statements that define Minimal Conformance and a set of optional features.

Minimal Conformance is the lowest level of conformance that can be claimed for XQuery. Minimal Conformance encompasses all XQuery functionality, with the exception of the following optional features:

- Schema Import Feature – allow the use of `import schema` in the prolog to make XQuery aware of the declarations of elements, attributes, and types.
- Schema Validation Feature – allows the use of the `validate` expression.
- Static Typing Feature – requires XQuery to detect and report type errors during the static analysis phase. Some queries that might run successfully without static typing will return an error during static analysis.
- Full Axis Feature – allows the use of the "reverse axes" `ancestor`, `ancestor-or-self`, `following`, `following-sibling`, `preceding`, and `preceding-sibling`.
- Module Feature – allows the use of `import module` in the prolog and allows library modules to be created.
- Serialization Feature – requires that an implementation provide a way to produce an XML serialization of the result of a query.
- Trivial XML Embedding Feature – allows an query to be provided as an XML element.

```
<xqx:xquery>for $e in ... </xqx:xquery>
```

Future Work

While we continue to move XQuery 1.0 through the W3C process towards its publication as a Recommendation, we have work underway that will add to XQuery 1.0.

Several Working Drafts (WD) have been published for XQuery 1.0 and Path 2.0 Full-Text [7]. Requirements have been published for an XQuery Update Facility [8], but an initial WD has not yet been published.

We expect that early next year the XML Query WG will begin considering features that could not be included in XQuery 1.0 for a future version of this Recommendation.

References

[1] *An Early Look at XQuery*, Andrew Eisenberg and Jim Melton, ACM SIGMOD Record, Vol. 31, No. 4, December 2002, http://www.sigmod.org/sigmod/record/issues/0212/AndrewEJimM.pdf.

[2] W3C XML Query (XQuery), http://www.w3.org/XML/Query/.

[3] W3C Technical Reports and Publications, http://www.w3.org/TR/.

[4] XML Query Test Suite, http://www.w3.org/XML/Query/test-suite/.

[5] *XML Schema Part 2: Datatypes Second Edition*, Paul V. Biron and Ashok Malhotra, Oct. 28, 2004, http://www.w3.org/TR/xmlschema-2/.

[6] *XSLT 2.0 and XQuery 1.0 Serialization*, Michael Kay, et al, Nov. 3, 2005, http://www.w3.org/TR/xslt-xquery-serialization/.

[7] *XQuery 1.0 and XPath 2.0 Full-Text*, Sihem Amer-Yahia, et al, Nov. 3, 2005, http://www.w3.org/TR/xquery-full-text/.

[8] *XQuery Update Facility Requirements*, Don Chamberlin and Jonathan Robie, June 3, 2005, http://www.w3.org/TR/xquery-update-requirements/.

[9] *XML Syntax for XQuery1.0*, Jim Melton and Subramanian Muralidhar, Nov. 3, 2005, http://www.w3.org/TR/xqueryx/.