An introduction to XQuery

A look at the W3C's proposed standard for an XML query language

Howard Katz

January 06, 2006
(First published June 01, 2001)

Howard Katz introduces the W3C's XQuery specification, currently winding its way toward Recommendation status. The complex specification currently consists of 15 separate working drafts and will likely grow some more before it's done. This article provides some background history, a road map into the documentation, and a snapshot of the current state of the specification. A sidebar takes a quick look at some key features of XQuery's surface syntax. Code samples demonstrate the difference between XQuery and XQueryX, and show examples of the surface syntax.

Note: Updates made to this article in December 2005 incorporate recent changes to the XQuery specification: Eight of the working drafts have now achieved W3C "Candidate Recommendation" status, bringing the specification as a whole much closer to final Recommendation. The main full-text document, first released in 2004, has recently been updated. A Requirements Working Draft for an update facility, as well as a draft on building an XPath/XQuery tokenizer, were both released for the first time in 2005. The number of XQuery features continues to grow, as does the list of XQuery implementers and the number of Web-based resources available to developers.

After six long years moving along the W3C's Recommendation track, the XQuery specification is taking on much of the mythic and enduring look of a Hollywood franchise -- "Star Wars" and the "Lord of the Rings" series come to mind. XQuery had its origins in a W3C-sponsored query language workshop held way back in 1998, in which representatives from industry, academia, and the research community gathered in Boston to present their views on the features and requirements they considered important in a query language for XML.

Two diverse constituencies

The 66 presentations, which are all available online for those interested in a historic perspective (see Related topics), came mainly from the members of two very distinct constituencies: those working primarily in the domain of XML as-document (largely reflecting XML's original roots in SGML), and those working with XML as-data -- the latter primarily reflecting XML's ever-increasing presence in the middleware realm, front-ending traditional relational databases.
One presentation in particular, a succinct and lucid presentation by David Maier of the Oregon Graduate Institute titled "Database Desiderata for an XML Query Language," particularly helped inform the thinking of the Query Language Working Group that was chartered shortly after Boston.

While its population has fluctuated somewhat over the years, the working group is large by W3C standards (I've been told that only the Protocol Working Group has a larger membership). Its composition of some 30-odd member companies reflects the views of both the data and the document constituencies. What's now coming close to coalescing into final form (at very long last) is an XML query language standard that ably manages to represent the needs and perspectives of both communities.

The key component of XQuery that will be most familiar to XML users is XPath, itself a W3C specification. A solitary XPath location path standing on its own (for example, "//book/editor" means "find all book editors in the current dataset") is perfectly valid XQuery. On the data side, XQuery's SQL-like appearance and capabilities should be both welcome and familiar to those coming in from the relational side of the world.

**Its humble origins**

XQuery started life as Quilt. Primarily a test vehicle for the user-level syntax, Quilt was spearheaded by three diligent and highly visible members of the working group: Jonathan Robie, Don Chamberlin, and Daniela Florescu. Quilt was in turn based on the collaborative efforts of the entire working group in defining requirements, use cases, and an underlying data model and algebra.

Robie, Chamberlin, and Florescu cite a number of language influences on Quilt's design, including XQL, XML-QL, and SQL. If you're interested in how computer languages evolve, read "XML Query Language: Experiences and Exemplars" (see Related topics), a useful paper that provides an excellent comparative overview of the first two languages, along with two others called YaTL and Lorel. The authors, Mary Fernandez, Jerome Simeon, and Phil Wadler, are themselves members of the working group.

Given the differing perspectives of the data versus document communities, and the solidity of the foundations being laid by the working group, it's not surprising that it took so long for the bulk of the specification to emerge into public view. For one thing, the inner proceedings of W3C working groups are confidential, and most of the Query Language Working Group's efforts prior to mid-February 2001 took place behind closed doors.

A Requirements document and a Data Model working draft were published fairly early on, but the working group's publishing arm really kicked into high gear in February 2001, when the bulk of the documentation started to appear. This was followed by two major updates in 2001, and three to four more updates annually ever since, with the exception of 2004, when the group only published once.

With the recent addition this year of a Requirements document for an update mechanism, as well as a shorter note on how to build a tokenizer for XQuery language implementers, the total is
now 16 documents (including the XSLT specification, listed on the XML Query Web site for some reason that's not clear to me), and that's likely to become a complete set before long. An update language document is sure to emerge at some point.

### A nascent publishing empire

The set of documents that in their totality describe and define XQuery currently consists of:

**XML Query Requirements**
- The main planning document for the working group. A list of XQuery desiderata.

**XML Query Use Cases**
- A number of real-world scenarios and XQuery snippets solving specific problems.

**XQuery 1.0: An XML Query Language**
- The central document, introducing the language itself and an overview of most everything else.

**XQuery 1.0 and XPath 2.0 Data Model**
- An extension of the XML infoset. Describes the data items that a query implementation must understand, and the basis of the formal semantics.

**XQuery 1.0 and XPath 2.0 Formal Semantics**
- The underlying algebra formally defining the language.

**XML Syntax for XQuery 1.0 (XQueryX)**
- An alternative syntax for those who prefer XML (primarily computers).

**XQuery 1.0 and XPath 2.0 Functions and Operators Version 1.0**
- Nearly 225 functions and operators on XML Schema datatypes, XML nodes, and sequences of both.

**XML Path Language (XPath) 2.0**
- The XPath documentation, broken out separately.

**XPath Requirements Version 2.0**
- The requirements document for XPath.

**XSLT 2.0 and XQuery 1.0 Serialization**
- A look at the considerations involved in outputting serialized *angle-bracket* XML from the XQuery 1.0 and XPath 2.0 Data Model. Serialization is not a part of the main language specification *per se*.

**XML Query and XPath Full-Text Requirements**
- A description of feature requests that a Full-Text Recommendation needs to be able to comply with.

**XML Query and XPath Full-Text Use Cases**
- Real-world scenarios that a Full-Text specification is expected to be able to handle.

**XQuery 1.0 and XPath 2.0 Full-Text**
- The main full-text document, detailing the full-text language extensions to XQuery proper.

**XQuery Update Facility Requirements**
- The features that XQuery requires to be able to write new data into existing documents, as well as query against them.
Building a Tokenizer for XPath or XQuery

A working draft note that breaks out and expands on some of the grammatical material originally found in the main XQuery 1.0 document. This would only be of interest to language implementers.

These documents (all referenced in Related topics) represent a prodigious body of work. XQuery 1.0: An XML Query Language is the linchpin document of the set, but the other documents all contribute toward what is an astonishingly well-specified and comprehensively backstopped language. To my knowledge this is the most complex set of specifications to come out of the W3C (although XML Schema arguably comes to mind, but that's another story ...).

If you're staring at this mass of documentation for the first time and wondering where to begin, I can recommend two possible approaches. You can start with the central XQuery 1.0 document. It has a good, introductory overview and details each of the language's many, many features. Another approach is to begin by picking up the Use Cases working draft. This document outlines a number of real-world scenarios where XQuery has applicability. Each use case targets a specific application domain and lists a number of XQueries posed against the sample data for that domain. The code snippets are invaluable if you like looking at concrete examples of real working syntax. A third approach, which works best if you already have a minimal understanding of the language, is to look through the many built-in functions listed in the Functions and Operators working draft.

Two excellent books have also appeared on the scene in the last several years to explain the ins and outs of the specification, both from Addison-Wesley: "XQuery from the Experts" presents a number of detailed technical essays on XQuery-related topics from members of the working group, while "XQuery: The XML Query Language" is an imminently readable reference work by Microsoft's Michael Brundage (see Related topics).

BabelFish, where are you?

XQuery is really three languages in one:

- The surface syntax is the most visible of the three and the one that users are most likely to come into contact with. For most purposes, this version of the language is XQuery. (See examples of the surface syntax in the sidebar Syntax: A quick sampler.)
- An alternative XML-based syntax replaces the surface language with one that's more tractable to machine processing. (See XQueryX, later in this article.)
- A formal algebraic language describes the inner workings of an XQuery processor in quite a bit of detail.

An underlying formalism

The Data Model and Formal Semantics working drafts together provide a precise, theoretical underpinning for XQuery. The two documents detail a query algebra, a set of precise definitions that define in formal terms the core entities that an XQuery query is expected to operate on, and formulations of what the various language operators can do with those operands. This likely won't be of interest to you unless you're a query-engine implementer, have major pocket protection, or simply like working with complex, formal systems.
One mapping that's provided enables implementers to recast surface syntax features directly into the underlying algebra. You can implement query processors that actually speak the algebra directly (although I would think this is more for proof-of-concept), as several vendors have demonstrated at XML trade shows. A link in Related topics points to an online demo version of one of these algebra-based engines.

The algebra also provides rules that detail how to optimize and transmute both complex expressions into simpler equivalents. As best I can tell (I'm not a language theorist, and the Formal Semantics document is far from light reading), both of these are good things. Large database vendors in particular will appreciate a query-language architecture that's designed from the ground up to be both optimizable and efficient.

The algebra also provides a place to hang type information. XQuery is strongly typed: If your data has a W3C XML Schema associated with it, a processor can validate against that schema and provide the query engine with Post-Schema Validation Infoset (PSVI) information about the datatypes of nodes in your documents, utilizing both types declared in "XML Schema Part 2: Datatypes" and user-defined types of your own. The algebra has both static and dynamic typechecking capability. For example, an engine can use PSVI-derived type information to statically check the datatype of query expressions at compile time (when the query's being analyzed for syntactic correctness). Determining that a query is type-invalid early in the cycle short circuits the need for doing potentially expensive (and fruitless) searches against large datasets. Much of the work on the XQuery specification has involved work on the syntax and semantics of the part of the language involving types.

**The transition to XPath 2.0**

XQuery shares a common data model with XPath 2.0, a fact reflected in the somewhat awkward title of the data model document: "XQuery 1.0 and XPath 2.0 Data Model" (and a reason that the Working Group has started using the much more pronounceable acronym, XDM, to refer to the data model). XPath 2.0 is just about fully baked at this point. The data model describes the core information in an XML document that's of interest to an XPath processor, and the final syntax and semantics of XPath's step operations is now almost completely worked out. The full specification is jointly owned by the Query Language and XSL working groups, and both groups need to concur on what XPath 2.0 will look like. At times that's been challenging, both politically and technically. However, if the road to consensus is sometimes a rocky one, both groups seem to be navigating it without too much obvious discomfort (at least as seen from an outsider's possibly naive perspective).

As just one example of why the transition from XPath 1.0 to 2.0 has been an interesting one, consider this: XPath 1.0 is a set-based expression language. Node-sets, one of the four datatypes in XPath 1.0, are just that: sets. By definition, sets are unordered and contain no duplicate members. XPath 2.0, on the other hand, is sequence based. By contrast, sequences of nodes in XPath 2.0 (not surprisingly, called node-sequences by analogy) have order, and duplicates are allowed. The ramifications of these differences were among a number of issues that had to be hammered out by the working groups separately and in concert as they brought themselves and XPath 2.0 into alignment, as the jargon goes.
Where are we now?

All of the substantive remaining issues have now been worked out. The fact that seven of the key documents comprising the XQuery spec are now Candidate Recommendations in W3C parlance means, in official terms, that XQuery is now considered "stable and appropriate for implementation."

In terms of the formal W3C Recommendation process, all issues raised during the previous Last Call period have been responded to, and the working group is now looking to industry vendors to provide real-life verification that XQuery's major features are implementable. To do this, implementers run their implementations through a test suite that's provided by the working group. Those features that aren't implemented by two or more vendors during the Candidate Recommendation stage are at risk of being dropped from the specification. The current list of at-risk features includes:

- Static typing
- Modules
- Collections
- Static typing
- Trivial XML embedding
- Copy-namespaces declaration

XQueryX

XQueryX, the specification of an alternative XML-based syntax for the surface language, was one of the earlier additions to the XQuery document family. One of the requirements for XQuery states that multiple syntaxes might be possible -- it sounds as if the working group was hedging its bets a bit -- and if so, one of these would have to be convenient for humans to read and write; the other would have to be expressible in XML. XQueryX is the working group's answer to the latter requirement.

Having an XML-based query representation has all the obvious, known advantages of XML: It makes it easy for standard tools to parse, generate, and interrogate the contents of a query. This might be useful, for example, if you're doing source-level optimization or transformation, which might depend in turn on the ability to easily inspect a query for a particular grammatical structure. XML is good at such tasks.

XQueryX is a near one-to-one mapping into XML of the formal grammar for the language. Given the complexity of the grammar, this makes XQueryX highly verbose, to the degree that it's nearly impossible for humans to read. Happily, machines -- which are the intended recipients of the language -- don't complain about such things. Listings 1 and 2 provide a comparison of a simple query expressed first in standard XQuery syntax and then in its XQueryX counterpart. Note the significant steroid-like bulk-up factor.
Listing 1. A simple query in standard syntax

```xml
<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>
```

Listing 2 shows the XQueryX equivalent. I've omitted about three-quarters of the listing due to its length. The full listing, lifted directly from the XQueryX working draft, runs to 132 lines:

Listing 2. The Listing 1 query in XQueryX format (snippet)

```xml
<?xml version="1.0"?>
<xqx:module xmlns:xqx="http://www.w3.org/2005/XQueryX" ... >
  <xqx:mainModule>
    <xqx:queryBody>
      <xqx:elementConstructor>
        <xqx:tagName>bib</xqx:tagName>
        <xqx:elementContent>
          <xqx:flworExpr>
            <xqx:forClause>
              <xqx:typedVariableBinding>
                <xqx:varName>b</xqx:varName>
              </xqx:typedVariableBinding>
              <xqx:forExpr>
                <xqx:pathExpr>
                  <xqx:argExpr>
                    <xqx:functionCallExpr>
                      <xqx:functionName>doc</xqx:functionName>
                      <xqx:arguments>
                        <xqx:stringConstantExpr>
                          http://bstore1.example.com/bib.xml
                        </xqx:stringConstantExpr>
                      </xqx:arguments>
                    </xqx:functionCallExpr>
                  </xqx:argExpr>
                </xqx:pathExpr>
              </xqx:forExpr>
              <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:forClauseItem>
          </xqx:flworExpr>
        </xqx:forClause>
        ...
      </xqx:elementContent>
    </xqx:queryBody>
  </xqx:mainModule>
</xqx:module>
```

All dressed up, and where do you go?

When I first wrote this summary of existing XQuery implementations in June 2001, just after the first major publishing iteration, only two implementations were available: a very early one of my own, and Microsoft's. That gave me an opportunity to poke some fun at myself, joking that Bill Gates and I were jockeying for market position. This time around, four years and any number
of working drafts later, that joke no longer works. Nearly four dozen implementations are now available, along with a large number of related products and tools.

The best place to see what’s currently available is on the XML Query home page (see Related topics). The list there is quite active, and I expect to see new implementations appearing on a regular basis, as interest and momentum build, and as the specification moves closer to Recommendation status.

Syntax: A quick sampler

Now I’ll take a quick look at a few XQuery features in the light of an actual example. Here’s a very simple query that operates on one of the canonical sample files in the Use Cases document. This query illustrates XQuery’s ability to both project (select a subset of nodes in the dataset that match desired criteria) and transform (produce an output document that differs from the one being queried against). XQuery allows you to both specify what you’re looking for and designate what its output format should look like in the same query.

Listing 3 shows a fragment of the document on which this query is operating:

Listing 3. Fragment of document that query is operating on

```xml
<bib>
  <book year="1994">
    <title>TCP/IP Illustrated</title>
    <author><last>Stevens</last><first>W.</first></author>
    <publisher>Addison-Wesley</publisher>
    <price>65.95</price>
  </book>
  <book year="1992">
    <title>Advanced Programming in the Unix environment</title>
    <author><last>Stevens</last><first>W.</first></author>
    <publisher>Addison-Wesley</publisher>
    <price>65.95</price>
  </book>
  <book year="2000">
    <title>Data on the Web</title>
    <author><last>Abiteboul</last><first>Serge</first></author>
    <author><last>Buneman</last><first>Peter</first></author>
    <author><last>Suciu</last><first>Dan</first></author>
  </book>
...
</bib>
```

You want the resulting output document (somewhat prettified) to look like Listing 4:
An introduction to XQuery

Listing 4. Resulting output document

```xml
<results>
    <book authorCount="1">
        <author>Stevens</author>
    </book>
    <book authorCount="1">
        <author>Stevens</author>
    </book>
    <book authorCount="3">
        <author>Abiteboul</author>
        <author>Buneman</author>
        <author>Suciu</author>
    </book>
    ...
</results>
```

And here's the query itself. Its job is to scan through all the books in the queried document, generating the result document shown above, which: contains a computed authorCount attribute in each new <book> tag being output; and discards most of the remaining information from the original, retaining only each author's last name.

(Note that I'm using the term "queried document" (singular) here. That's a simplification: XQuery's data model is also capable of handling collections of documents, as well as partial fragments.)

Listing 5. Query to scan all books in queried document

```xml
<results>
        let $authors := $book/author
        return
        <book authorCount="{ count($authors) }">
            { for $author in $authors
                return
                <author>{ $author/last/text() }</author>
            }
        </book>
    }
</results>
```

In Listing 5, the doc() function is used to point the query at the XML document being interrogated. It returns a document node in the vocabulary of the XQuery and XPath Data Model (XDM).

Dissecting the query

Here are a few interesting features of this query:

**for/let expressions**

The example contains two nested for loops and a let. The outer for iterates through each of the nodes resulting from the expansion of the path expression, doc(...)//book, isolating each <book> node in turn in a variable named $book. The let expression in turn picks up all the <author> subnodes of each book in a variable named $authors. The $authors variable holds a node sequence; the $book and $author variables both hold single nodes.
It's important to note that these variables aren't *assigned to*, they're *bound*. The distinction is subtle but important: Once a variable has been bound, its value is immutable. This prevents nasty side effects that can result from reassigning the value of a variable on the fly. Another potential benefit is that lines containing variables can (to some degree) be rearranged during processing, allowing savvy engines to optimize their queries.

The `for` and `let` expressions are subcomponents of a *FLWOR* (pronounced *flower*) expression. The acronym stands for its five major component clauses: `for`-`let`-`while`-`order by`-`return`. In *Listing 6*, the formal grammar for a *FLWOR* expression:

**Listing 6. FLWOR expression with for and let subcomponents**

```
FLWORExpr ::= (ForClause | LetClause)+ WhereClause? "return" Expr
```

shows that it's quite a protean expression type, capable of generating a large number of possible query instances. As this production shows, the `Expr` term following the "return" keyword can itself be replaced by another *FLWOR* expression, so that *FLWOR*’s can be strung together on end *ad infinitum*, like an ever-lengthening sequence of LEGO blocks. The replacement of an `Expr` term by any other expression type is what makes XQuery *composable* and gives it its rich, expressive power. There are a large number of expression types in XQuery, each capable of being plugged into the grammar wherever a more generic `Expr` is called for.

On a more mundane note, eventually a `return` statement terminates a *FLWOR* sequence. And in the case of the query above, an additional internal `return` is used as a convenient point to insert an element constructor for each `<book>` that's being output.

**Element constructors**

The query contains three element constructors. The elements `<results>`, `<book>`, and `<author>` are generated on the fly by writing the literal angle-bracket XML directly into the body of the query itself.

Braces (`{ and }`) are used where necessary to disambiguate literal text content from subexpressions inside an element constructor that require evaluation. For example, if you were emitting the literal expression in *Listing 7*, braces wouldn't be required to separate the inner and outer tags.

**Listing 7. Code without braces to separate inner and outer tags**

```
<authors>
  <author>
    ...
```

Braces, by the way, were introduced in the June 2001 revision of the surface-language syntax. Earlier versions of the grammar didn't require them. Braces are a good example of how the language changes and evolves as it moves toward Recommendation.

**Attribute constructors**
The code in **Listing 8** shows the use of an inline attribute constructor. The `count()` function returns the number of `<author>` elements contained in each book. Note again the braces, used here to cordon off an expression requiring evaluation from its surrounding literal XML. The use of quotes to delimit attribute values in attribute constructors is an example of another change that was made to the specification as it evolved.

**Listing 8. Use of an inline attribute constructor**

```
<book authorCount="{ count($authors) }" />
```

**Built-in functions and operators**

`count()` is an example of a built-in function. The "Functions and Operators" draft lists close to 225 functions and operators in some dozen different groups that construct and operate on a wide variety of datatypes, including numbers, strings, booleans, dates and times, qnames, nodes, and sequences.

The expression in **Listing 9** uses the `text()` operator to populate the contents of each `<author>` element, with the text of the last name pulled out of its enclosing `<last>` element. If you just used `$author/last` directly, you'd get the enclosing tag as well, something that's not desired in this case.

**Listing 9. Use of the built-in operator, text()**

```
<author>{ $author/last/text() }</author>
```
Related topics

- **XML Query Requirements**: The planning document for the working group; a list of XQuery desiderata.
- **XML Query Use Cases**: A number of real-world scenarios and XQuery snippets solving specific problems.
- **XQuery 1.0: An XML Query Language**: The keystone document, detailing the surface language and an overview of just about everything else.
- **XQuery 1.0 and XPath 2.0 Data Model**: An extension of the XML infoset. It describes the data items that a query implementation needs to deal with. Now being called the XDM.
- **XQuery 1.0 and XPath 2.0 Formal Semantics**: The underlying algebra formally defining query engine behavior.
- **XML Syntax for XQuery 1.0 (XQueryX)**: An alternative XML syntax for queries, preferred by machines.
- **XML Path Language (XPath) 2.0**: The XPath 2.0 specification, broken out separately from XQuery.
- **XQuery 1.0 and XPath 2.0 Functions and Operators**: Roughly 225 of them.
- **XSLT 2.0 and XQuery 1.0 Serialization**: The official word on how to produce angle-bracket XML.
- **XQuery and XPath Full-Text Requirements**: What's demanded of an XQuery full-text sub-language.
- **XQuery and XPath Full-Text Use Cases**: Real-world scenarios for full-text.
- **XQuery 1.0 and XPath 2.0 Full-Text**: Full-text language extensions to XQuery proper.
- **XPath Requirements Version 2.0**: The requirements document for XPath.
- **XQuery Update Facility Requirements**: The features that XQuery requires to be able to write new data into existing documents, as well as query against them.
- **Building a Tokenizer for XPath or XQuery**: A working draft note that breaks out and expands on some of the grammatical material originally found in the main XQuery 1.0 document. This would only be of interest to language implementers.
- **XML Query Languages: Experiences and Exemplars**: This paper identifies essential features of an XML query language by examining four existing query languages: XML-QL, YATL, Lorel, and XQL.
- **XML Schema Part 2: Datatypes**: Datatypes understood by XQuery.
- **QL98: Query Languages Workshop**: W3C Web site for the original Boston workshop on query languages.
- **Database Desiderata for an XML Query Language**: David Maier’s presentation to the workshop.
- **XML Query Language: A Proposal**, Robie, Lapp, and Schach: The XQL proposal to the W3C.
- **Quilt**: Robie, Florescu, and Chamberlin’s Web site for this XQuery precursor.
- "**XQuery from the Experts**" (Addison-Wesley, 2003): A compendium of technical essays on XQuery by members of the XML Query working group, edited by yours truly.
• **Galax**: Jerome Simeon's formal-semantics-based query engine demo at Bell Labs. "Formal semantics" is now the official term for what was called "algebra" prior to the June 7, 2001 release.

• **XQEngine**: The author's own open-source, Java-based query engine. It's now somewhat out of date, as the author has, at least for the moment, given up on trying to keep up with recent changes in the spec.

• **Mark Logic Content Server**: Leading book publishers (among others) use this package. You can store and query 50 MB of content for free.

• **Berkeley DB XML**: Sleepycat's open-source native XML database, based on their venerable and highly scalable Berkeley DB database engine.

• IBM's **DB2** database provides relational database storage, plus pureXML to quickly serve data and reduce your work in the management of XML data.