An Early Look at XQuery API for Java™ (XQJ)

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

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

Introduction

In Feb. 2004, the period for submitting Last Call Working Draft comments for most parts of the XQuery specification came to a close. While there is still a great deal of work to be done to make XQuery a W3C Recommendation, the documents have become more stable with each public release. In this column we'd like to provide an initial look at the XQuery API for Java™ (XQJ), a project that is taking place within the Java Community Process (JCP).

You might find it surprising that this project, which is so closely related to XQuery, is not owned by the XML Query Work Group (WG), or at least by a WG within the W3C. The XML Query WG has written XQuery to be largely independent of the environment in which it is used. It has deliberately left the creation of APIs to other groups.

The embedding of XQuery within another environment is worthy of mention. SQL/XML:2003 [1] defined an XML data type for SQL and it defined publishing functions that generate XML values from relational data. INCITS H2 and ISO's SC32/WG3 are currently working on features that would allow the evaluation of XQuery expressions within SQL and the transformation of XQuery result sequences to virtual tables.

Java Community Process (JCP)

For those of you not familiar with JCP [2], let's look at the first item in this organization's FAQ:

> Q: What is the JCP?
>
> A: Since its introduction in 1998 as the open, participative process to develop and revise the Java™ technology specifications, reference implementations, and test suites, the Java Community Process (JCP) program has fostered the evolution of the Java platform in cooperation with the international Java developer community.

Members of this organization, which is owned by Sun Microsystems, are free to submit Java Specification Requests (JSRs). Once accepted, the Specification Lead (Spec Lead) forms an Expert Group (EG) from among the JCP members and outside individuals, and the EG begins its work. In some cases, the role of Spec Lead is filled by multiple companies. The Spec Lead is responsible for producing:

Specification: The specification should be complete enough to allow for the creation of an independent implementation of this technology.

Reference Implementation (RI): An implementation of the Specification. A "proof of concept".

Technology Compatibility Kit (TCK): A test suite that is used to determine whether an implementation has correctly implemented the Specification. The RI must pass the TCK. Independent implementations of the Specification must also pass the TCK.

The Spec Lead determines the terms and conditions of the licenses for these three deliverables of the JSR.

JSR 225, XQuery API for Java (XQJ)

In May 2003, we jointly submitted the JSR for XQJ to JCP on behalf of our employers, IBM and Oracle. It was designated JSR 225 and was accepted shortly thereafter [3]. The description we provided for the JSR says, in part:

> "This specification will define a set of interfaces and classes that enable an application to submit XQuery queries to an XML data source and process the results of these queries. The design of the API will also take into account precedents established by other JSRs, notably JDBC and JAXP.
>
> SQL (developed by INCITS H2 and ISO/IEC JTC 1/SC 32/WG 3) is the query language supported by many relational DBMSs. JDBC is the Java API that allows an application to submit SQL requests to an RDBMS and process the results of the query. This specification relates to XQuery in the same way that JDBC relates to SQL."

The two of us have become the Spec Leads for this JSR. IBM and Oracle have agreed to jointly produce the specification for XQJ, with Oracle producing the RI and IBM producing the TCK. Our companies have said that we will offer the Specification, RI, and TCK on a Royalty-Free basis, with commonly-used disclaimers and warranties on the technologies.

We formed the XQJ Expert Group in June 2003 and expect to release an Early Draft Review of this JSR in May 2004. This Early Draft Review provides the first visibility of this work to the public.

Relevant Aspects of XQuery

As we have said, XQuery is part way through its progression to a W3C Recommendation. The latest versions of the specification can be obtained from the W3C web site [4]. You might also want to look at an earlier article that we wrote on XQuery [6].

For the purposes of designing and discussing XQJ, some aspects of XQuery are more relevant than others. The XQuery Data Model defines the atomic values and nodes that are provided as input to evaluation, are intermediate products of evaluation, and are the results of evaluation. The XQuery Data Model supports heterogeneous sequences of items that may be either atomic values or nodes. XQuery's process model defines both a static analysis phase and a dynamic evaluation phase for the processing of an expression.

XQuery has an initial static context that is used when its static analysis is performed. The static context contains items like statically known namespaces, type definitions, element and attribute declarations, in-scope variables (names and types), and in-scope functions. The initial values for these items are determined by the XQuery implementation, subject to some constraints. These items can be augmented (and in some cases overwritten) by the XQuery prolog. The static analysis depends on the expression and the static context, but not on input data.

XQuery has an initial dynamic context that is used when the dynamic evaluation is performed. The dynamic context contains the values of variables defined in the static context and the values associated with known documents and collections.

The XQuery prolog can contain variable declarations that specify "external". Values for these variables must be provided by the environment, in this case XQJ, before the expression can be evaluated.

XQJ and JDBC

In designing XQJ, we have been very cognizant of the effort that has gone into JDBC (its fourth version is now being developed). Where JDBC allows SQL statements to be executed, we must allow XQuery expressions to be evaluated. SQL has a number of connection properties that can be set with SQL statements. XQuery has the static context, which can be set by prolog directives.

JDBC allows an SQL statement to be prepared, dynamic parameter markers (specified by "?") bound, and then executed. XQJ will allow this as well, with external variables taking the place of dynamic parameter markers.

Where SQL produces tables as the result of execution, XQuery produces sequences of items. The rows of SQL tables are homogenous, while the items in an XQuery sequence are heterogeneous.

Because SQL's Information Schema tables have not been broadly adopted, JDBC provides a number of metadata methods such as `DatabaseMetaData.getTables()`. XQJ defines its own metadata methods, and also defines XQuery functions to provide access to the metadata associated with documents, collections, modules, etc.

Writing Applications with XQJ

This API is being designed to allow an application to:

- establish a connection to an XML data source
- determine some of the properties of the data source
- discover the persistent objects that may be referenced in an XQuery expression
- prepare an XQuery expression
- execute a prepared XQuery expression with values that have been bound to external variables
- operate on the sequence of items that the execution produces

We will say more about each of these features in turn.

As you would expect, XQL allows an application to close connections, expressions, etc. in order to free up resources. It also allows applications to handle exceptions and provide useful information to the user of these applications. We are not going to say any more about these somewhat mundane features in this article.

Establish a Connection to an XML Data Source

An application begins its interaction with the XQJ API by either creating or obtaining an object that implements the `XQDataSource` interface (perhaps

via Java Naming and Directory Interface™, JNDI, in a J2EE environment). In order to create this object, an application will have to load a class that has been obtained from a supplier of this technology. This supplier will likely provide a JAR file containing classes that implement all of the XQJ interfaces.

The following example shows the case where the application knows both the name of the class that it will use, and the properties that it will set.

```
XQDataSource ds =
  (XQDataSource)
    Class.forName("com.acme.xml.XQDS")
    .newInstance();
ds.setProperty("serverName",
               "www.example.com");
ds.setProperty("portNumber", "8888");
ds.setLoginTimeout(15);
```

Instances of other XQJ interfaces are created, either directly or indirectly, from the `XQDataSource` object.

The XML data source might be local to the machine that is executing the application or it might be remote. The XML data source might be a file system containing XML documents, an XML database, or an XML mapping of an SQL database. XQJ only requires that the XQuery Data Model and XQuery language be supported by the XML data source.

The application can now create one or more connections using this data source.

```
// provide name and password
String passwd = ... ;
XQConnection con =
   ds.getConnection("Jim", passwd);
```

`XQConnection` provides set and get methods for the properties of updatability, holdability, and scrollability (forward only or scrollable). The expressions that are created from a connection will use the most recently set values of these properties. Further thought is needed on how the updatability and holdability properties that were taken from JDBC might apply to XQJ.

Determine Properties of the Data Source

The `XQMetaData` interface provides a number of methods that an application can use to discover the properties of the XML data source. These properties fall into a number of categories:

- Product identification
- XQJ specification identification
- Connection information
- User information
- Product capabilities
- XQuery supported features
- Product limits

An application might use some of these methods as follows:

```
XQMetaData md = con.getMetaData();

// display product name
System.out.println("Product: "
                   + md.getProductName());
```

XQJ also allows an application to get these properties directly from the connection.

```
String xqjVProp
   = "javax.xml.xquery.metadata.XQJVersion";
String xqjSIProp
   =   " javax.xml.xquery.metadata"
     + ".SchemaImportSupported";

// display the version of XQJ
System.out.println("Version: "
      + con.getMetaDataProperty(xqjVProp));

// display whether Schema Import is
// supported
System.out.println("Schema Import: "
      + con.getMetaDataProperty(xqjSIProp));
```

Applications do not need both of these methods of discovering data source properties. The Expert Group will likely choose one of them and remove the other in the future.

Static and Dynamic Context

XQJ provides a number of methods that allow an application to examine parts of the XQuery static context. It does not provide methods to alter it, relying instead on the ability of applications to write prolog directives into their queries. These methods are provided by the `XQConnection` interface (because this interface implements the `XQStaticContext` interface).

The opposite is true of the methods for the XQuery dynamic context. These methods allow an application to alter its state, but not to examine these values. For the most part, the application will bind values to variables in the dynamic context. These values will be seen by variable references in the query. An application can also get and set the value of the XQuery implicit timezone.

The application can change the dynamic context via methods of the `XQConnection` interface. These changes are seen by all expressions and prepared expressions that are subsequently created. The application may also change the dynamic context with corresponding methods on the `XQExpression` and `XQPreparedExpression` interfaces. These methods change the dynamic context locally.

An application might bind a document or a sequence of document nodes to a variable globally,

and then prepare and then execute an XQuery repeatedly, binding values to a string or an integer variable. XQJ allows both the global and local variables to be bound to any instance of XQuery Data Model, which includes atomic values, such as strings and integers, and nodes (document nodes, element nodes, etc.).

In order for a global variable to be used, it must be declared as an external variable in the XQuery prolog. Without this declaration, the variable will not be visible to the expression, and a reference to it will raise an error. The following example shows how this can be done:

```
XQConnection con = ... ;
org.w3c.dom.Node myNode = ... ;
String myQuery
   = "declare variable $sales external; "
     + "count($sales//salesperson)";

con.bindNode(new QName("sales"), myNode);
XQResultSequence rs =
   con.createExpression()
      .executeQuery(myQuery);
```

Prepare and Execute an XQuery

In the following example, we will prepare an XQuery expression that will count the employees of a given department. We will then execute the query with departments “Accounting” and “Facilities”.

```
String xquery1 =
   "declare variable $name "
 + "  as xs:string external; "
 + "for $d in doc('depts.xml')/depts/dept "
 + "where $d/@name = $name "
 + "return count($d/employees/employee)";

XQPreparedExpression expr1
   = con.prepareExpression(xquery1);

expr1.bindString(new QName("name"),
                 "Accounting");
XQResultSequence rs = expr1.executeQuery();
rs.next();
System.out.println(rs.getInt());
rs.close();

expr1.bindString(new QName("name"),
                 "Facilities");
rs = expr1.executeQuery();
rs.next();
System.out.println(rs.getInt());
rs.close();
```

The execution of the `prepareExpression` method causes XQuery’s static analysis to take place.

Once a query has been prepared, an application might want to examine the static type that has been inferred for the result of the query. XQJ provides the `XQSequenceType` interface and the `getStaticResultType` method for this purpose. We’re not going to provide an example of this because, quite frankly, this area needs some improvement.

The one-time execution of a query is also supported.

```
String xquery2 =
   "for $d in doc('depts.xml')/depts/dept "
 + "where $d/location/floor = 5 "
 + "return data($d/@name)";

XQExpression expr2
   = con.createExpression();
XQResultSequence rs
   = expr2.executeQuery(xquery2);
while (rs.next()) {
   System.out.println(rs.getString());
}
rs.close();
```

The `executeQuery` method allows XQuery expressions to be evaluated. `XQExpression` also supports the `executeCommand(String)` method. The language accepted by this method is determined by the XML data source. This method might allow an application to process administrative or DDL statements.

We have used XQuery in the examples of XQJ that we have provided. XQJ also supports the use of XQueryX [9], the XML representation of XQuery, in preparing and executing expressions.

Operate on the Resulting Sequence of Items

An application can choose to operate on the result of the query, an instance of the `XQResultSequence` interface, in two ways. The application can operate on the entire sequence all at once, or it can operate on it one item at a time.

The `getSequenceAsString` and `writeSequence` methods allow an application to retrieve a serialized XML result into a Java object or a file. These methods accept an argument that contains serialization parameters that can influence just how this serial representation is produced.

XQJ provides access to nodes via the DOM, SAX, and StAX interfaces. Methods such as `writeSequenceToSAX` and `writeSequenceToStream` provide this access via SAX and StAX, respectively.

Operating on the Items Individually

In our design of this aspect of XQJ, we were vividly aware that a result could have a very large number of result items, and that these result items could be as small as an integer, or as large as an entire document. Therefore, we provide item-at-a-time access to the result sequence.

A result sequence is initially positioned before the first item. An application can traverse the result sequence one item at a time, using the `next` method. The `getItemType` method can be used to determine the kind of value on which the cursor is positioned. Numerous "get" methods, such as `getInt` and `getString` are provided to create a Java value or object from the value of the item.

A result sequence can be requested to be either scrollable or forward only. A scrollable result sequence allows the application to invoke a number of methods to test the position of the cursor within the sequence and to change the position with methods such as `first()`, `last()`, `previous()`, and `relative(int)`.

The result sequence is valid only as long as the expression that created it has not been closed or has not been used to process another query. If the result sequence is forward only, then once the value of the current item has been retrieved, it cannot be retrieved again without re-executing the query.

Instead of retrieving the value directly, an application can invoke the `getItem` method that returns an instance of the `XQResultItem` interface. This object is valid for as long as the expression and sequence that created it have not been closed or used to process another query. This object can then be used to retrieve the value of the item.

The result of a query might be displayed as follows:

```
XQResultSequence rs = ... ;
XQDataSource ds = ... ;
int n = 0;
XQItemType itype;
XQItemType atomic
   = ds.createItemType
        (XQItemType.XQITEMTYPE_ATOMIC,
         null,
         null,
         false);

while (rs.next()) {
   System.out.println("item " + n++ + ":");
   itype = rs.getItemType();
   System.out.println("   type: "
      + itype.getString());
   if (itype.isOfType(atomic))
      System.out.println("   value: "
         + rs.getLexicalValue());
   else
      System.out.println("   node: "
         + rs.getNode().getNodeName());
}
```

The `getLexicalValue` method returns a string value for any atomic value. The `getNode` method returns a DOM node, an instance of `org.w3c.dom.Node`.

Another Use of these Results

The result of a query is an object that implements the `XQResultSequence` interface. A specific item in the sequence, obtained with the `getItem` method, is an object that implements the `XQResultItem` interface. Both of these can be used to bind their value to a variable, which can be used in a subsequent query. This subsequent query may use the same connection as the first, or a different connection. If different connections are used, they may connect to different XML data sources and those connections may even be provided by different vendors.

Retrieving a Node

An item that is a node may be accessed in the following ways:

DOM	`getNode()`
SAX	`writeSAX (org.xml.sax.ContentHandler)`
StAX	`getXMLStreamReader()` or `writeXMLStream (javax.xml.stream .XMLStreamWriter)`

Other XML object models, such as DOM4J, JDOM, or JAXB, are not supported directly. Recognizing that some applications will want to use these XML object models, or other models that have not been anticipated, the `XQCommonHandler` interface has been provided. This interface has just two methods, one that takes an argument of `java.lang.object` and produces an `XQItem`, and another that takes an argument of `XQItemAccessor` (a super-interface of `XQItem` and `XQSequence`) and returns a `java.lang.object`.

A user or organization can implement this interface and then use it to invoke the `getObject` method as follows:

```
// Convert the nodes to Employee objects
XQCommonHandler handler = new myHandler();
XQResultSequence rs = expr.executeQuery();
while (rs.next())
{
   Employee e
      = (Employee) rs.getObject(handler);
}
```

An application may also use the `XQConnection.setCommonHandler` method to set a common handler for use by all expressions and result sequences. The `getObject()` method uses the common handler that has been set in such a way. The `getObject(XQCommonHandler)` method allows a specific handler to be used, overriding the one that was set for the connection.

Support of Ad-Hoc Applications

An application might accept a query from a file or from a user via a graphical user interface of some type. It could then prompt the user for values of the external variables, execute the query, and finally display the results.

To this end, the `XQPreparedExpression` interface provides two methods, `getAllExternalVariables` and `getAllUnboundVariables`, both of which return an array of QNames. This interface also provides the `getStaticVariableType` method, which takes a QName and returns an instance of the `XQSequenceType` method. For example:

```
XQPreparedExpression expr = ... ;
QName[] qn
   = expr.getAllUnboundExternalVariables();
for (int i = 0; i < qn.length; i++) {
   // Display variable
   ...expr.getStaticVariableType(qn[i])... ;
   // Prompt for a value
   // bind the variable
}
```

Connection-Independent Sequences

The results of queries that we have seen, instances of the `XQResultSequence` interface, are valid only as long as the expression that created them has not been closed. An application may create instances of the `XQCachedSequence` interface that can remain valid until the application itself ends.

The following example shows how such a sequence can be created and used to bind to a variable.

```
XQDataSource ds = ... ;
String xquery3 = ... ;
XQItemType xs_string
   = con.createItemType
        (XQItemType.XQITEMTYPE_ATOMIC,
         null,
         new QName
               ("integer","xsi",
                "http://www.w3.org/2001/"
                 + "XMLSchema-instance"),
         false);
XQCachedSequence seq1 =
   ds.createCachedSequence();
seq1.insertString
   ("Accounting", xs_string);
XQResultSequence rs
   = con.createExpression()
        .executeQuery(xquery3);
```

The result sequence will then be used to create a sequence that survives beyond even the connection that created it.

```
XQCachedSequence seq2
   = ds.createCachedSequence();
seq2.insertSequence(rs);
con.close();

// seq2 is valid  after connection is closed
while (seq2.next())
   { ... }
```

XQJ also allows an application to create instances of the `XQCachedItem` interface. These objects are like the instances of `XQResultItem`, but have a lifetime that is independent of any connections.

`XQResultSequence` and `XQCachedSequence` are both sub-interfaces of `XQSequence`. Many methods are defined using `XQSequence`, and so accept instances of either of the sub-interfaces. A similar statement can be made for `XQResultItem`, `XQCachedItem`, and `XQItem`, respectively.

Examining Metadata

Some applications have a need to discover the objects that exist on an XML data source so that they can construct valid XQuery expressions. Let's look at these types of objects and the XQuery syntactic constructs that operate on them.

schema	import schema prolog directive
module	import module prolog directive
collation	order by portion of FLWOR expression, string functions such as fn:contains
function	function invocation
collection	fn:collection accessor function
document	fn:doc accessor function

Either URIs or QNames are used to identify these types of objects. An XQuery implementation might have a "closed world" view and only allow the use of objects contained by the XML data source or those that have been registered in some way. An implementation might instead have an "open world" view and resolve any references that it receives and then attempt to dynamically load these objects.

XQJ specifies that an XML data source must provide a number of XQuery functions to access metadata, an XML schema describing the nodes returned by these functions, and two namespace prefixes in its initial static contexts (`xqjm`, to identify the metadata schema, and `xqj-util`, to identify the metadata functions).

Primary among these functions is the `metadata` function, which returns a document with elements that represent all of the types of objects that we discussed. This document could be directly returned to an application, or it could be used as input

to other XQuery expressions that filter, project, and aggregate its contents.

The following query would return an element for each function that is defined with 4 or more parameters:

```
for $f in xqj-util:metadata()
          /metadata/functions/function
where count($f/parameters/param) >= 4
return $f
```

An implementation might choose to extend the XQJ metadata schema to provide more information about the kinds of objects that we have mentioned, or to provide information about entirely new kinds of objects.

XQuery says very little about documents and collections. The two accessor functions, `fn:collection` and `fn:doc`, accept a URI and return a sequence of nodes and a document node, respectively. XQJ supports both implementations that have flat collections and those that support nested collections (collections that can have other collection children as well as document children). An implementation can determine whether nested collections are supported by examining the XML data source properties. The XQJ metadata schema allows an implementation to reflect the contents of nested collections if they are supported.

XQJ also specifies a number of XQuery helper functions that provide access to specific portions of this metadata. Some of these functions return just the URIs of documents or collections, rather than their complete description.

The following query returns an element for each module that contains a function with a local name of “soundex”.

```
for $m in xqj-util:available-modules
where $m/functions/function/@localName
      = "soundex"
return $m
```

Future Work

There’s very little that we can say about exactly how this work will proceed … we’re limited by the member confidentiality of the Expert Group. We’ve provided a very early view of this project, and we freely admit that even the existing functionality needs a great deal of refinement.

The issue of typing requires a great deal of additional work. We need to decide how XQuery’s type annotations are made visible to an application. Also, the static type of an expression, inferred by XQuery/XPath Formal Semantics [8], is more precise and expressive those that can be expressed by XQuery’s SequenceType production. Whether this rich type info needs to be exposed via the XQJ API and how it might be provided needs more thought.

Skipping over the intermediate milestones that XQuery and XQJ must each achieve, our goal is to bring XQJ to Final Release soon after XQuery becomes a W3C Recommendation.

Acknowledgments

This JSR is being progressed with the help of the members of the XQJ Expert Group. Two individuals deserve special acknowledgement: Jan-Eike Michels (IBM) and Muralidhar Krishnaprasad (Oracle) have been serving as editors of our Specification and our API (Javadoc), respectively.

References

[1] *ISO/IEC 9075-14:2003, Information technology - Database languages - SQL - Part 14: XML-Related Specifications (SQL/XML)*, International Organization for Standardization, Geneva, 2003.

[2] *The Java Community Process(SM) Program*, http://www.jcp.org.

[3] *The Java Community Process(SM) Program — JSRs: Java Specification Requests — detail JSR# 225*, http://www.jcp.org/en/jsr/detail?id=225.

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

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

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

[7] *XQuery 1.0: An XML Query Language*, Scott Boag, Don Chamberlin, Mary F. Fernández, Daniela Florescu, Jonathan Robie, Jérôme Siméon, Nov. 12, 2004, http://www.w3.org/TR/xquery/.

[8] *XQuery 1.0 and XPath 2.0 Formal Semantics*, Ashok Malhotra, Kristoffer Rose, Michael Rys, Jérôme Siméon, Philip Wadler, Denise Draper, Peter Fankhauser, Mary Fernández, Feb. 20, 2004, http://www.w3.org/TR/xquery-semantics/.

[9] *XML Syntax for XQuery 1.0 (XQueryX)*, Michael Rys, Ashok Malhotra, Jim Melton, Jonathan Robie, Dec. 19, 2003, http://www.w3.org/TR/xqueryx.