CentraSite

The CentraSite API for XQJ

Version 9.5 SP1

November 2013

This document applies to CentraSite Version 9.5 SP1.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Document ID: IINM-DG-XQJ-95SP1-20140410

Table of Contents

The CentraSite API for XQJ

You can use XQJ, the XQuery API for Java™, for processing XML and for data integration applications. This document introduces the CentraSite implementation of XQJ and its features. It explains how to use the API and provides examples for each type of task.

The reader of the document should be an experienced Java programmer.

This document covers the following topics.

What is XQJ?	Introduces the XQuery API for Java and explains its features
Working with the CentraSite XQJ Interface	Describes various tasks you can perform with XQJ
CentraSite-Specific Extensions to XQJ	Describes proprietary facilities for updating a database and inserting documents in the registry/repository.
XQDataSource Properties	Lists and explains all datasource properties
Javadoc: XQJ Extensions	Javadoc for CentraSite-specific extensions to the XQJ API.
Messages and Codes: XQJ messages	Messages and codes listed

1 What is XQJ?

XQJ, the XQuery API for Java, is based on XQuery, a query language promulgated by the W3C that can operate both on physical XML documents, and also on virtual XML documents that have been derived from data sources such as relational or object databases. XQJ is a powerful new API standard developed for invoking XQuery expressions against virtually any XML or relational database and processing query results. XQJ makes the full power of the XQuery language available to Java applications. You can programmatically process the results in your Java code in a JDBC-like manner. XQJ is to XQuery what JDBC is to SQL.

The XQJ standard specifies a number of Java interfaces. The CentraSite XQJ interface implements the functionality defined by these interfaces, and thus makes XQJ available to the application; in addition, the CentraSite XQJ interface implements extensions that support CentraSite-specific features.

Note: With effect from version 8.2, CentraSite supports the final release of the XQJ specification (in contrast, earlier versions of CentraSite supported a preliminary release of the XQJ specification). Note that the XQJ interface that is implemented by current versions of CentraSite is not compatible with the interface that was implemented by versions of CentraSite prior to version 8.2. Documentation of the prior XQJ interface is available to Software AG customers who have a current maintenance contract in Empower, Software AG's global extranet (**http://empower.softwareag.com/**).

Features of the CentraSite XQJ Interface

The CentraSite XQJ interface supports:

- Prepared XQueries
- The submission of queries to the CentraSite registry/repository
- XQuery updates
- Transaction control (commit, rollback)
- User authentication prior to connecting to the database
- Variable binding to parameterize queries
- Handling registry/repository errors and warnings
- The creation and execution of materialized sequences and items
- Different models for accessing data in the CentraSite registry/repository (DOM, SAX, and StAX-compatible streams)

2 Working with the CentraSite XQJ Interface

If you wish to develop an XQJ application, you will find the classes of the CentraSite XQJ implementation in the jar file *rts/bin/xqj.jar* under the CentraSite installation location.

You can use the CentraSite XQJ interface to perform an XQuery on the basis of a standard XQExpression or an XQPreparedExpression. With a standard XQExpression, the query is parsed each time it is executed. If a query is to be executed many times, it can be more efficient to use an XQPreparedExpression, which is parsed only once.

Executing an XQuery with a Standard XQExpression

To execute an XQuery with a standard XQExpression

1 Invoke the `getXQConnection()` method to get the XQConnection object from the JAXRConnection.

Example

```
/* Get the XQConnection from the JAXRConnection */
XQConnection connection = jaxrConnection.getXQConnection ();
```

You have now established an XQConnection.

2 Create an XQExpression object from the XQConnection object. The XQExpression is used to invoke several other methods to perform various tasks using the CentraSite XQJ interface. You may create more than one XQExpression from a single connection if required.

Example

```
/* Create XQExpression from XQConnection to execute an XQuery. */
XQExpression expression = connection.createExpression();
```

3 Optionally, you can bind one or more external variables. An external variable is a type of variable that can be dynamically added to the query by declaring the variable in the query. The value of the variable can be set externally and added to the pre-set variable while executing the XQuery.

Example

```
String xquery = "declare variable $year as xs:int external" +
                "for $q in input()/bib/book where $q/@year > $year return $q" ;
XQExpression expression = connection.createExpression();
expression.bindInt(new QName("year"),1993,XQItemTypeHelper.createIntXQItemType());
XQResultSequence xqResultSequence = expression.executeQuery(xquery);
```

4 Invoke the `executeQuery()` method. This returns an XQResultSequence.

Example

```
/* Executing an XQuery */
/* Instance of the query string: */
String xquery = "for $b in input()/book return $b/title";
/* Execute the above XQuery String, which returns an XQResultSequence */
XQResultSequence xqResultSequence = expression.executeQuery(xquery);
```

5 The XQResultSequence represents the XQuery result. Retrieve the query result and read/print it in XML format. The query result sequence is displayed item by item. Using XQJ, it is possible to get the result sequence in DOM, SAX and StAX-compatible formats.

Note: You cannot scroll the XQResultSequences backwards.

Example

```
/* Iterating the XQResultSequence */
XMLStreamReader reader = null;
While(xqResultSequence.next())
{
  reader =  xqResultSequence.getItemAsStream();
  /* Iterate the XML StreamReader using StAX-compatible APIs */
}
connection.commit();
connection.close();
```

Example using the getInt() method

```
/* Instance of the XQuery String */
String xquery =" for $b in input()/bib/book return xs:int($b/@year) ";
/* This query on execution will return the year as an integer value */
XQResultSequence xqResultSequence = expression.executeQuery(xquery);

xqResultSequence.next();
int I = xqResultSequence.getInt();
```

Example using the getAtomicValue() method

```
/* Instance of the XQuery String */
String xquery = "for $p in input()/book return xs:string($p/title)";
XQResultSequence xqResultSequence = expression.executeQuery(xquery);

/* This query on execution will return the title as a String */
xqResultSequence.next();
String str = xqResultSequence.getAtomicValue();
```

Example using the getNode() method

```
/* Instance of the XQuery String */
String xquery = "for $q in input()/bib/book return $q";
XQResultSequence xqResultSequence = expression.executeQuery(xquery);
xqResultSequence.next();
Node node = result.getNode();
```

Example using the writeItemToSAX() method

```
xqResultSequence.next();
StringWriter sw = new StringWriter();

/* Provide a org.xml.sax.ContentHandler, which is saxhandler in our case */
XQSAXTextEventHandler saxhandler = new XQSAXTextEventHandler(sw);
resultSequence.writeItemToSAX(saxhandler);
System.out.println(sw);
```

6 Finally, invoke the `XQConnection.close()` method to close the connection to the registry/repository.

Example

```
/* Commit and close the XQConnection once you have completed working with it */
connection.commit();
connection.close();
```

Executing an XQuery with an XQPreparedExpression

To execute an XQuery with an XQPreparedExpression

1 Invoke the `getXQConnection()` method to get the XQConnection object from the JAXRConnection.

Example

```
/* Get the XQConnection from the JAXRConnection */
XQConnection connection = jaxrConnection.getXQConnection ();
```

You have now established an XQConnection.

2 Create an XQPreparedExpression object from the XQConnection object. The XQPreparedExpression is used to invoke several other methods to perform various tasks using the CentraSite XQJ interface. You may create more than one XQPreparedExpression from a single connection if required.

Example

```
/* Create XQPreparedExpression from XQConnection */
String pQuery = "for $q in input()/bib/book return $q";
XQPreparedExpression preparedExpression = conn.prepareExpression(pQuery);
```

3 Optionally, you can bind one or more external variables. An external variable is a type of variable that can be dynamically added to the query by declaring the variable in the query. The value of the variable can be set externally and added to the pre-set variable while executing the XQuery.

Example

```
/* Binding variables in Prepared Expressions */
String pQuery = "declare variable $int as xs:int external" +
                "for $q in input()/bib/book where $q/@year = $int return $q";

XQPreparedExpression preparedExpression = conn.prepareExpression(pQuery);

/* Bind the appropriate value to the prepared expression using the matching ↩
binding API provided. */
```

Using bindInt() to bind an int value to the prepared expression

```
preparedExpression.bindInt(new QName("int"), 1994, ↩
XQItemTypeHelper.createIntXQItemType());
```

Using bindNode() to bind a node to the prepared expression

```
/* Get a node to bind by executing an expression */
XQExpression expression = connection.createExpression();
XQResultSequence xqResultSequence =
    expression.executeQuery("for $q in input()/bib/book where $q/@year = 1994 ↩
return $q/title");
xqResultSequence.next();

/* Get a node from the result sequence retrieved above */
Node node = xqResultSequence.getNode();

/* PreparedQuery */
String pquery = "declare variable $node external " +
                "for $q in input()/bib/book where $q/title = $node return $q";
XQPreparedExpression prepared = connection.prepareExpression(pQuery);

/* Bind the above retrieved node to the prepared query */
prepared.bindNode(new QName("node"), node);
```

4 Invoke the `executeQuery()` method. This returns an XQResultSequence.

Example

```
/* Execute the prepared expression which returns an XQResultSequence */
XQResultSequence xqResultSequence = preparedExpression.executeQuery();
```

5 The XQResultSequence represents the XQuery result. Retrieve the query result and read/print it in XML format. The query result sequence is displayed item by item. Using XQJ, it is possible to get the result sequence in DOM, SAX and StAX-compatible formats.

Example

```
/* Iterating the XQResultSequence */
XQResultSequence xqResultSequence = preparedExpression.executeQuery();
XMLStreamReader reader = null;
While(xqResultSequence.next())
{
  reader =  xqResultSequence.getItemAsStream();
  /* Iterate the XML StreamReader using StAX-compatible APIs */
}
```

Example using the getInt() method

```
XQResultSequence xqResultSequence = preparedExpression.executeQuery();
xqResultSequence.next();
int I = xqResultSequence.getInt();
```

Example using the getAtomicValue() method

```
XQResultSequence xqResultSequence = preparedExpression.executeQuery();
xqResultSequence.next();
String str = xqResultSequence.getAtomicValue();
```

Example using the getNode() method

```
XQResultSequence xqResultSequence = preparedExpression.executeQuery();
xqResultSequence.next();
Node node = result.getNode();
```

Example using the writeItemToSAX() method

```
XQResultSequence xqResultSequence = preparedExpression.executeQuery();
xqResultSequence.next();
StringWriter sw = new StringWriter();

/* Provide an org.xml.sax.ContentHandler, which is saxhandler in our case */
XQSAXTextEventHandler saxhandler = new XQSAXTextEventHandler(sw);
resultSequence.writeItemToSAX(saxhandler);
System.out.println(sw);
```

6 Finally, invoke the `XQConnection.close()` method to close the connection to the registry/repository.

Example

```
/* Commit and close the XQConnection once you have completed working with it */
connection.commit();
connection.close();
```

Working with a Materialized XQSequence

A materialized sequence is not bound to any connection or XQuery expression. It can be created from XQResultSequences or from a java.util.iterator.

Examples

Creating a Sequence

This example demonstrates how to create a materialized sequence from Java collection via the java.util.iterator interface. It creates a materialized sequence holding 3 int items.

```
ArrayList items = new ArrayList();
items.add(conn.createItemFromInt(123,null));
items.add(conn.createItemFromInt(456,null));
items.add(conn.createItemFromInt(789,null));
XQSequence sequence = conn.createSequence(items.iterator());
```

Creating a Copy from an XQResultSequence

This example demonstrates how an XQResultSequence can be copied into a materialized sequence. The materialized sequence will exist independently of the XQResultSequence.

```
String query = "for $q in input()/bib/book where $q/@year = 1994 return $q";
XQExpression expression = connection.createExpression();
XQResultSequence resultSequence = expression.executeQuery(query);

XQSequence Sequence = connection.createSequence(resultSequence);
```

3 CentraSite-Specific Extensions to XQJ

Software AG's CentraSite adds useful facilities to the XQJ interface for updating a database and for inserting a document into the CentraSite registry/repository. These facilities are described in the following sections.

Updating a Database Using XQJ

Using the CentraSite XQJ interface, you can update the registry/repository. This feature is a Software AG specific extension of XQJ.

To update an XQuery

1 Specify the string or the reader object containing the update XQuery

2 Invoke the `executeUpdate()` method on the expression

Example

```
String updateQuery = "update for $q in input()/bib/book where $q/@year = 1994" +
"do replace $q/title with <title>XQJ from SoftwareAG </title>";
XQResultSequence xqResultSequence = ↩
((XQExpressionImpl)expression).executeUpdate(updateQuery);
// execute update
```

Inserting a Document in the CentraSite Registry/Repository

This feature is a Software AG specific extension of XQJ.

To insert a document in the CentraSite Registry/Repository

1 Specify the XML instance to be inserted as a string or the reader object.

2 Execute the `executeInsert()` method in XQExpression to insert a document.

Example

```
String insertStr = "<your xml goes here>";
(XQExpressionImpl)expression).executeInsert(insertStr);
```

A XQDataSource Properties

In addition to the standard properties, CentraSite offers the following properties for parameterizing XQJ connections. Note that user credentials, i.e. user-ID and password, are passed via standard properties, as shown below:

Standard Properties

Property	Description
`javax.xml.xquery.property.UserName`	Unique user ID for connecting to the registry/repository.
`javax.xml.xquery.property.Password`	The password for the specified user ID.
`javax.xml.xquery.property.MaxConnections`	The maximum number of open connections that can be established from the datasource.

CentraSite-Specific Properties

Property	Description
`com.softwareag.tamino.xqj.dbUri`	Mandatory. The URI of the database to which the user is connecting. This information is mandatory to connect to the datasource, which is the CentraSite registry/repository in this context.
`com.softwareag.tamino.xqj.defaultCollection`	Mandatory. The name of the collection in the registry/repository that the user will access to query, update, or insert a document.
`com.softwareag.tamino.xql.locale`	The locale to be set for the connection.
`com.softwareag.tamino.xqj.isolationLevel`	Together with the `_lockMode` parameter, this parameter specifies the way in which two or more transactions in a session context can access the same data simultaneously. The isolation level can be set to "None".

com.softwareag.tamino.xqj.lockMode	Together with the _isolationLevel parameter, this parameter specifies the way in which two or more transactions in a session context can access the same data simultaneously.
com.softwareag.tamino.xqj.lockWait	The action to be taken if data is not accessible to the current transaction because another transaction has used the _isolationLevel or _lockMode parameter to restrict access to the data.
com.softwareag.tamino.xqj.fetchSize	The number of records to be retrieved at a time for display. This property accepts an integer value.
com.softwareag.tamino.xqj.sensitive	The parameter _sensitive is required when opening a cursor with _xquery. Valid values are "no" and "vague". If you specify "_sensitive=no", an insensitive cursor is opened. This means that the query is calculated on a fixed input when the cursor is opened, and thus the result sequence remains unchanged as long as the cursor is active. If you specify "_sensitive=vague", a vague cursor is opened. The query is calculated on an input that takes modification operations of parallel transactions into account. Thus, the result sequence can vary during the lifetime of the cursor if documents that match the original query criteria are inserted, updated or deleted in the meantime.
com.softwareag.tamino.xqj.nonactivityTimeout	The non-activity timeout in seconds.