

Work with GPX XML in DB2 9.5 using JDBC

Administration and application programming

Skill Level: Intermediate

[Khurram Faraaz \(khfaraaz@in.ibm.com\)](mailto:khfaraaz@in.ibm.com)
Staff Software Engineer
IBM

[Ronny Bartsch \(ronny.bartsch@gmx.net\)](mailto:ronny.bartsch@gmx.net)
Software Developer

[Susan Malaika \(malaika@us.ibm.com\)](mailto:malaika@us.ibm.com)
Senior Technical Staff Member
IBM

15 Jan 2009

Many XML capabilities were introduced in IBM® DB2® 9 and 9.5 through the pureXML™ feature. In this article, see how you can exercise administrative functions, such as XML metadata management, and application development functions, such as XML manipulation and storage, through JDBC.

Introduction

Java™ technology and JDBC are popular technologies for accessing and manipulating stored data. With the introduction of the XML data type in DB2 9, new Java programming techniques are needed. The latest JDBC driver, JDBC 4.0, supports the XML data type. This article provides examples of registering XML schema as well as storing, retrieving, querying, transforming, and removing XML data from DB2 using the JDBC 4.0 driver.

This article explains how to work with XML data in DB2 9.5 using JDBC. Learn how to perform both [administrative tasks](#) and [application development tasks](#).

The XML notation used in the article is called GPX (GPS Exchange Format). It is used for representing GPS (Global Positioning Systems) data.

Administrative tasks overview

In this article, you will create the artifacts needed to store and manipulate XML data. Furthermore, you will also populate a table with XML data through DB2 supplied utilities. The tasks described include:

- From a script:
 - Create a table 'GPX' that has both relational columns as well as an XML column
 - Create a stored procedure from the DB2 command prompt that queries the table 'GPX' and is later called from a Java program
- From a Java program:
 - Create XML indexes on an XML column in the 'GPX' table
 - Drop the previously CLP-registered XML schema, then register the GPX XML schema in the XML Schema Repository again to enable import of XML instances with validation in a later step
 - Import XML data from a file into the 'GPX' table using a DB2 supplied stored procedure and the DB2 `IMPORT` utility
 - Export data (both XML and relational) from 'GPX' table to external files using a DB2-supplied stored procedure and the DB2 `EXPORT` utility

Application development tasks overview

In this article, you will also insert, retrieve, modify, and transform XML from a Java program using JDBC APIs. In particular, you will:

- Insert data into both relational and XML columns into the 'GPX' table using an `insert` statement and an external XML file
- Insert XML data with validation against the registered GPX XML schema into the 'GPX' table using an XML data string
- Retrieve data from both relational and XML columns from the 'GPX' table (this article provides examples of SQL statements and XQuery statements being executed)

- Call and execute a stored procedure that is already registered in DB2
- Insert XML data with validation against a given XML schema
- Retrieve and transform GPX data using XQuery by inserting an element into an XML document, modifying the value of an existing XML element in an XML document, and deleting an XML element
- Retrieve and transform stored GPX data through XSLT to produce HTML

Prerequisites

You need to have the following software installed on your Windows system:

- [DB2 Express-C](#) or DB2 9.5 LUW
- JDK 1.6 or later

Before running any of the supplied JDBC samples (see [Download](#)), you need to establish a connection to the data source, as described in the following section. The scripts and article assume that all the work is being done directly on the C: drive.

Establish a connection to a data source using the JDBC DriverManager

A JDBC application can establish a connection to a data source using the JDBC DriverManager interface, which is part of the `java.sql` package.

The Java application first loads the JDBC driver by invoking the `Class.forName` method. After the application loads the driver, it connects to a database server by invoking the `DriverManager.getConnection` method.

For the IBM DB2 Driver for JDBC and SQLJ, load the driver by invoking the `Class.forName` method with the following argument:
`com.ibm.db2.jcc.DB2Driver`

Load the IBM DB2 Driver for JDBC and SQLJ

To create a connection to a database using the IBM DB2 Driver for JDBC, you must provide three parameters:

- A user ID
- The related user password
- A URL that defines the location of the database

The URL consists of four parts with the following meanings:

- **jdbc:db2:** Indicates that the connection is to a DB2 for z/OS or DB2 for Linux, UNIX, and Windows server
- **server:** The host name or IP address of the database server
- **port:** The TCP/IP server port number that is assigned to the database server (this is an integer between 0 and 65535; the default is 50000)
- **database name:** The name that was defined during creation of the database (in the article example, it is "gpx")

Listing 1 demonstrates loading the IBM DB2 Driver for JDBC and SQLJ:

Listing 1. Loading the IBM DB2 Driver for JDBC and SQLJ

```
try {
    // Load the DB2 JDBC Type 4 Driver with DriverManager
    Class.forName ("com.ibm.db2.jcc.DB2Driver");
} catch (ClassNotFoundException e) {
    e.printStackTrace ();
}
checkForCommandLineArgs (s);
try {
    Connection con = null;
    String url = "jdbc:db2://" + server + ":" + portNo + "/" + dbName;
    con = DriverManager.getConnection (url, userName, password);
```

The `catch` block is used to print an error if the driver is not found.

Administrative tasks

Configure environment

In this section, you will set up an environment that is required by the Java programs used in this article. Execute the script `GPX_configure_environment.db2` (see [Download](#)) from the DB2 command prompt before trying to execute any of the sample programs provided with this article (see [Download](#)):

```
C:\>db2cmd
C:\>db2 -td@ -vf GPX_configure_environment.db2
```

The `GPX_configure_environment.db2` script file includes the following statements, shown in Listing 2:

Listing 2. Configure environment

```
-- Create a database in UTF-8 codeset and territory US.
CREATE DATABASE GPX USING CODESET UTF-8 TERRITORY US
-- Connect to the GPX database.
```

```

CONNECT TO GPX
-- Create schema GPXADMIN.
-- Create table GPXADMIN.GPX
create table GPXADMIN.GPX(ID int not null primary key,
                          COMMENT varchar(1000) not null, DOCUMENT xml)
-- Register the XML Schema GPX.xsd in DB2's XSR as GPXADMIN.GPX
-- Complete the registered XML Schema.
-- Import data into the GPX table using the IMPORT utility.
IMPORT FROM ./GPX_BaseSampledata.del OF DEL XML FROM ./sampledata
XMLVALIDATE USING XDS INSERT INTO GPXADMIN.GPX
-- Set the current schema as GPXADMIN.
-- Create procedure GPXADMIN.QUERY1 with this definition,

CREATE PROCEDURE GPXADMIN.QUERY1(IN sym_param VARCHAR(10))
SPECIFIC QUERY1
DYNAMIC RESULT SETS 1
LANGUAGE SQL
BEGIN

DECLARE SQLCODE INTEGER;
DECLARE stmt_text VARCHAR (4096);
DECLARE stmt STATEMENT;
DECLARE curl CURSOR WITH RETURN FOR stmt;

SET stmt_text='XQuery declare default element namespace
"http://www.topografix.com/GPX/1/1" ; for $y in
db2-fn:xmlcolumn("GPXADMIN.GPX.DOCUMENT")/gpx/wpt[sym="' || sym_param || '"]
return <type>{$y}</type>';

PREPARE stmt FROM stmt_text;
OPEN curl;

END@

```

Create database objects (table, XML index)

In this section, you will create a table that has both a relational column and an XML column, which is required by the Java programs used in this article.

Use the following file from the included [download](#): CreatedBObjects.java.

To execute this Java file, execute the following two commands:

```

C:\>javac CreatedBObjects.java
C:\>java CreatedBObjects

```

Listing 3 shows what is done in the above Java program:

Listing 3. Create database objects (table, XML index)

```

// 1. Load the DB2 JDBC Type 4 Driver with DriverManager.
Class.forName("com.ibm.db2.jcc.DB2Driver");

// 2. Create a Connection object to establish a connection to DB2
//    using the getConnection method.

static String server="localhost";
static int portNo=50000;
static String dbName="gpx";
static String userName="db2admin";

```

```
static String password="db2admin";

Connection con = null;
String url = "jdbc:db2://" + server + ":" + portNo + "/" + dbName;

con = DriverManager.getConnection (url, userName, password);

// 3. Prepare a set schema statement.

String setSchema = "SET CURRENT SCHEMA GPXADMIN";
PreparedStatement setSchStmt = con.prepareStatement(setSchema);

// 4. Execute the prepared statement.

setSchStmt.execute();

// 5. Prepare and execute create index statements.

String crtXMLIdx01 = "create index GPXADMIN.xmlIndex1 on GPXADMIN.GPX(DOCUMENT)
GENERATE KEYS USING XMLPATTERN \'declare default element namespace
\'http://www.topografix.com/GPX/1/1\' ; /gpx/wpt/sym\' as sql varchar(40)";

PreparedStatement crtIdxstmt01 = con.prepareStatement(crtXMLIdx01);
crtIdxstmt01.execute();

String crtXMLIdx02 = "create index GPXADMIN.xmlIndex2 on GPXADMIN.GPX(DOCUMENT)
GENERATE KEYS USING XMLPATTERN \'declare default element namespace
\'http://www.topografix.com/GPX/1/1\' ; /gpx/wpt/type\' as sql varchar(40)";

PreparedStatement crtIdxstmt02 = con.prepareStatement(crtXMLIdx02);
crtIdxstmt02.execute();

String crtXMLIdx03 = "create index GPXADMIN.xmlIndex3 on GPXADMIN.GPX(DOCUMENT)
generate key using xmlpattern \'declare default element namespace
\'http://www.topografix.com/GPX/1/1\' ; /gpx/wpt/@lon\' as sql DOUBLE";

PreparedStatement crtIdxstmt03 = con.prepareStatement(crtXMLIdx03);
crtIdxstmt03.execute();

String crtXMLIdx04 = "create index GPXADMIN.xmlIndex4 on GPXADMIN.GPX(DOCUMENT)
generate key using xmlpattern \'declare default element namespace
\'http://www.topografix.com/GPX/1/1\' ; /gpx/wpt/@lat\' as sql DOUBLE";

PreparedStatement crtIdxstmt04 = con.prepareStatement(crtXMLIdx04);
crtIdxstmt04.execute();
```

Register XML schema and import data

In this section, you will drop a previously registered XML schema and register an XML schema document in DB2's XSR (XML Schema repository) from a Java program. You will complete the XML schema registration process. Please note that the `drop` command is executed at the beginning of the `RegisterXMLSchema.java` program to remove the previously registered schema using CLP. See how to import data into a table using DB2's `import` utility that is executed from the `ADMIN_CMD` stored procedure; this step is optional, as the required data is already imported into the table using the `GPX_configure_environment.db2` script.

Use the following file from the included [download](#): `RegisterXMLSchema.java`.

To execute this Java file, execute the following two commands:

```
C:\>javac RegisterXMLSchema.java
```

```
C:\>java RegisterXMLSchema
```

Listing 4. Registration of an XML schema, import using ADMIN_CMD stored procedure

```
// 1. Load the DB2 JDBC Type 4 Driver with DriverManager.
Class.forName("com.ibm.db2.jcc.DB2Driver");

// 2. Create a Connection object to establish a connection to DB2
// using the getConnection method.

static String server="localhost";
static int portNo=50000;
static String dbName="gpx";
static String userName="db2admin";
static String password="db2admin";

Connection con = null;
String url = "jdbc:db2://" + server + ":" + portNo + "/" + dbName;

con = DriverManager.getConnection (url, userName, password);

// 3. Set the current schema to GPXADMIN using the prepareStatement method.

String setSchema = "SET CURRENT SCHEMA GPXADMIN";
PreparedStatement setSchStmt = con.prepareStatement(setSchema);
setSchStmt.execute();

// 4. REGISTER XML Schema in the XML Schema repository,
// using the SYSPROC.XSR_REGISTER stored procedure.

CallableStatement cstmt = con.prepareCall(
"CALL SYSPROC.XSR_REGISTER (?, ?, ?, ?, ?)");

String xsrObjectName = "gpx";
String xmlSchemaLocation = "gpx.xsd";

File schemaDoc = new File("./schemas/gpx.xsd");
InputStream inputStrm = new FileInputStream(schemaDoc);

long length = schemaDoc.length();

cstmt.setString(1,"GPXADMIN");

cstmt.setString(2, xsrObjectName);
cstmt.setString(3, xmlSchemaLocation);
cstmt.setBinaryStream(4, inputStrm,(int)length);
cstmt.setNull(5, java.sql.Types.BLOB);
cstmt.registerOutParameter(1, java.sql.Types.VARCHAR);
cstmt.registerOutParameter(2, java.sql.Types.VARCHAR);
cstmt.execute();

// 5. Complete the XML Schema using the SYSPROC.XSR_COMPLETE stored procedure.

CallableStatement completeXSRstmt =
con.prepareCall("CALL SYSPROC.XSR_COMPLETE (?, ?, ?, ?)");
completeXSRstmt.setString(1,"GPXADMIN");
completeXSRstmt.setString(2,"GPX");
completeXSRstmt.setNull(3,java.sql.Types.VARCHAR);
completeXSRstmt.setInt(4,0);

completeXSRstmt.execute();
completeXSRstmt.close();

// 6. Verify that the XML Schema is registered and completed successfully in the XSR.
```

```

String selFromXSR = "select * from sysibm.sysxsrobjects where XSROBJECTNAME='GPX'";
PreparedStatement pstmt = con.prepareStatement(selFromXSR);
pstmt.execute();

ResultSet res = pstmt.executeQuery();

while (res.next())
{
    String data = res.getString("XSROBJECTNAME");
    System.out.println(data);
}

// 7. This demonstrates how to import data into a table from a java program using the
// SYSPROC.ADMIN_CMD stored procedure which in turn uses the DB2 import utility.

// This is optional; following lines of code are commented in RegisterXMLSchema.java,
// as the import command is executed from GPX_configure_environment.db2.

String importStmt = "IMPORT FROM \"C:/sampledata/GPX_BaseSampledata.del\"
OF DEL XML FROM \"C:/sampledata\" XMLVALIDATE USING XDS INSERT INTO GPXADMIN.GPX";

CallableStatement impStmt = con.prepareCall(
    "call SYSPROC.ADMIN_CMD(?)");
impStmt.setString(1,importStmt);
impStmt.execute();

// 8. To drop an xsrobject from the XSR, this will drop GPXADMIN.GPX from the XSR;
// you will have to register the XML Schema again if required.

String dropXSR = "DROP XSROBJECT GPXADMIN.GPX";
PreparedStatement dropStmt = con.prepareStatement(dropXSR);
dropStmt.execute();

```

Export data to an external file using EXPORT command

In this section, you will export data from a table to external files. Note that the output from the export is not well-formed XML, as there is no root element and there are interspersed XML processing instructions. The data will be exported into two files: C:\data (without file extension), which is a delimited ASCII file containing references and offsets to access the full set of XML data stored in C:\data.001.xml, the second file.

Use the following file from the included [download](#): ExportXMLData.java.

To execute this Java file, execute the following two commands:

```
C:\>javac ExportXMLData.java
```

```
C:\>java ExportXMLData
```

Listing 5. Export data to an external file using EXPORT command

```

// 1. Follow the steps mentioned in any of the previous steps to load DB2
// JDBC Type 4 Driver and establish a connection to DB2.

// 2. Create a statement object using the connection object.
Statement stmt = con.createStatement();

// 3. Exports data from a database to one of several external file formats.

```

```
// The user specifies the data to be exported by supplying an SQL SELECT
// statement, or by providing hierarchical information for typed tables.
// The data is exported to the server only.

CallableStatement cstmt = con.prepareCall("CALL SYSPROC.ADMIN_CMD (?)");

String exportCMD = "export to c:\\data of DEL select * from GPXADMIN.GPX";

cstmt.setString(1,exportCMD);
cstmt.execute();
```

Application development tasks

Insert XML data into table

In this section, you will insert XML data into an XML column in a table. See how to retrieve the inserted XML data using the method `getSQLXML()` from object of type `SQLXML`. Please note that you need JDK 1.6 or later and JDBC 4.0 or later in order to use the `SQLXML` data type.

If you get an unresolved class `SQLXML` error, then please ensure you have the Java 1.6 compiler installed. Type `javac -version` to check the version.

Use the following file from the included [download](#): `InsertXML.java`.

To execute this Java file, execute the following two commands:

```
C:\>javac InsertXML.java
C:\>java InsertXML
```

Listing 6. Insert XML data into table

```
1. import java.sql.SQLXML;

// 2. Load the DB2 JDBC Type 4 Driver with DriverManager.
Class.forName("com.ibm.db2.jcc.DB2Driver");

/*
3. Create a Connection object to establish a connection to DB2
using the getConnection method.
*/

static String server="localhost";
static int portNo=50000;
static String dbName="gpx";
static String userName="db2admin";
static String password="db2admin";

Connection con = null;
String url = "jdbc:db2://" + server + ":" + portNo + "/" + dbName;

con = DriverManager.getConnection (url, userName, password);

// 4. Create a statement object.
Statement stmt = con.createStatement();
```

```

/*
5. Prepare and execute an insert statement.
   Insert XML data from a file input as binary data
*/

String insrtStmt = "insert into GPXADMIN.GPX values(?,?,?)";

PreparedStatement pstmt = con.prepareStatement(insrtStmt);
pstmt.setInt(1,5000);

pstmt.setString(2,"Bangalore");

File binFile = new File("Bangalore.xml");
InputStream inBin = new FileInputStream(binFile);
pstmt.setBinaryStream(3,inBin,(int)binFile.length());

pstmt.execute();

// 6. Retrieve data (XML and Relational) using resultSet.getXXX methods.

String sql2 = "select * from GPXADMIN.GPX where id = ?";
PreparedStatement pstmt2 = con.prepareStatement(sql2);

pstmt2.setInt(1,5000);
ResultSet rs2 = pstmt2.executeQuery();

while (rs2.next ()) {
    SQLXML xmldata = rs2.getSQLXML("DOCUMENT");
    System.out.println (rs2.getString ("id") + " " + xmldata.getString());
}

```

Insert XML data with validation against a given XML Schema

Here, you will insert XML data with validation against a given XML schema that is already registered in DB2's XSR.

Use the following file from the included [download](#): InsertXMLWithValidation.java.

To execute this Java file, execute the following two commands:

```

C:\>javac InsertXMLWithValidation.java
C:\>java InsertXMLWithValidation

```

Listing 7. Insert XML data with validation against a given XML schema

```

/*
1. Follow the steps mentioned in any of the previous steps to load
   DB2 JDBC Type 4 Driver and establish a connection to DB2.
*/

// 2. Create a statement object using the connection object.
Statement stmt = con.createStatement();

// 3. Prepare an insert statement with validation and execute it.

//To insert XML Document into XML column with validation against a given XML Schema.

String insrtStmt = "insert into GPXADMIN.GPX values(?,?,XMLVALIDATE(
XMLPARSE(DOCUMENT(CAST(? as VARCHAR(10000))) strip whitespace)
according to xmlschema id gpxadmin.gpx))";

PreparedStatement pstmt = con.prepareStatement(insrtStmt);

```

```

/*
 Please make sure that the value passed to parameter one is different each time
 you execute this program, as there is a primary key defined on the ID column.
*/

 pstmt.setInt(1,12345);

 pstmt.setString(2,"Germany");

 // XML data is assigned to string variable insrtstr.

 String insrtStr = "
 <?xml version=\"1.0\" encoding=\"UTF-8\"?>
 <gpx xmlns=\"http://www.topografix.com/GPX/1/1\" creator=\"byHand\" version=\"1.1\"
 xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"
 xsi:schemaLocation=\"http://www.topografix.com/GPX/1/1 ..\\schemas\\gpx.xsd\">
   <wpt lat=\"32.921055008\" lon=\"3.014223107\">
     <ele>11.862281</ele>
     <time>2006-05-16T11:49:06Z</time>
     <name>XYZ Location</name>
     <sym>City</sym>
   </wpt>
 </gpx> ";

 pstmt.setString(3,insrtStr);
 pstmt.execute();

```

Retrieve XML data from XML column

Now, you will retrieve XML data from a table using different APIs.

Use the following file from the included [download](#): RetrieveXML.java.

To execute this Java file, execute the following two commands:

```
C:\>javac RetrieveXML.java
```

```
C:\>java RetrieveXML
```

Listing 8. Retrieve XML data from XML column

```

1. import java.sql.SQLXML;

// 2. Load the DB2 JDBC Type 4 Driver with DriverManager.

Class.forName("com.ibm.db2.jcc.DB2Driver");

/*
3. Create a Connection object to establish a connection to DB2
using the getConnection method.
*/

static String server="localhost";
static int portNo=50000;
static String dbName="gpx";
static String userName="db2admin";
static String password="db2admin";

Connection con = null;
String url = "jdbc:db2://" + server + ":" + portNo + "/" + dbName;

con = DriverManager.getConnection (url, userName, password);

```

```

// 4. Prepare and execute a select statement.

String selStmt = "SELECT ID, DOCUMENT from GPXADMIN.GPX where ID = ?";
PreparedStatement pstmt = con.prepareStatement(selStmt);
pstmt.setInt(1,4);

/*
5. One way to retrieve data is, to execute the above prepared statement using the
executeQuery method, and then use the resultSet.getString(column-name) method.
*/

ResultSet resultSet = pstmt.executeQuery();
while(resultSet.next())
{
    System.out.println(resultSet.getString("DOCUMENT"));
}

/*
6. Another way would be to stream an XML value and use the
getBinaryStream(String columnName) method, is like this.
*/

resultSet.next();

InputStream inputStream = resultSet.getBinaryStream("DOCUMENT");

int c;
System.out.println(" The following data is retrieved using
    resultSet.getBinaryStream(DOCUMENT) ");

while((c=inputStream.read())!= -1)
{
    System.out.print((char)c);
}

// 7. Another way to retrieve data is to use an object of type SQLXML.

ResultSet res = pstmt.executeQuery();

res.next();
SQLXML xmldata = res.getSQLXML("DOCUMENT");
System.out.println(xmldata.getString());

// 8. You can also use a Reader object to retrieve data.

ResultSet resultSet = pstmt.executeQuery();
Reader reader = resultSet.getCharacterStream("DOCUMENT");
do{
    System.out.print((char)reader.read());
}
while(reader.read()!=-1);

```

Execute XQuery statement

In this section, you will execute an XQuery from a Java program. The query returns waypoints from GPX documents, which contain "Dam" in the element "gpx/wpt/sym".

Use the following file from the included [download](#): ExecXQuery.java.

To execute this Java file, execute the following two commands:

```
C:\>javac ExecXQuery.java
```

```
C:\>java ExecXQuery
```

Listing 9. Execute XQuery statement

```
// 1. Load the DB2 JDBC Type 4 Driver with DriverManager.
Class.forName("com.ibm.db2.jcc.DB2Driver");

/*
2. Create a Connection object to establish a connection to DB2
using the getConnection method.
*/

static String server="localhost";
static int portNo=50000;
static String dbName="gpx";
static String userName="db2admin";
static String password="db2admin";

Connection con = null;
String url = "jdbc:db2://" + server + ":" + portNo + "/" + dbName;

con = DriverManager.getConnection (url, userName, password);

// 3. Prepare and execute an XQuery statement.

String xQuery = "xquery declare default element namespace
\"http://www.topografix.com/GPX/1/1\" ; for $y in db2-fn:xmlcolumn" +
"('GPXADMIN.GPX.DOCUMENT')/gpx/wpt[sym=\"Dam\"] " +
"return <type> { $y } </type>";

PreparedStatement selectStmt = con.prepareStatement(xQuery);
ResultSet rs = selectStmt.executeQuery();

// Print the results of the XQuery using the result set object.
while(rs.next() )
{
    System.out.println(rs.getString(1));
}
```

Call stored procedure from Java program

Let's now call a stored procedure from a Java program. The stored procedure was installed in the "[Configure environment](#)" section and executes a query similar to the one in the previous section.

Use the following file from the included [download](#): StoredProcedureCall.java.

To execute this Java file, execute the following two commands:

```
C:\>javac StoredProcedureCall.java
C:\>java StoredProcedureCall
```

Listing 10. Call stored procedure from Java program

```
// 1. Load the DB2 JDBC Type 4 Driver with DriverManager.
Class.forName("com.ibm.db2.jcc.DB2Driver");
```

```

/*
2. Create a Connection object to establish a connection to DB2
using the getConnection() method.
*/

static String server="localhost";
static int portNo=50000;
static String dbName="gpx";
static String userName="db2admin";
static String password="db2admin";

Connection con = null;
String url = "jdbc:db2://" + server + ":" + portNo + "/" + dbName;

con = DriverManager.getConnection (url, userName, password);

/*
3. This is to demonstrate how to call a stored procedure.
Stored procedure GPXADMIN.QUERY1 is registered in DB2 from the CLP using a
script file GPX_configure_environment.db2.
*/

CallableStatement clstmt = con.prepareCall("call GPXADMIN.QUERY1(?)");
clstmt.setString(1, "Dam");
clstmt.execute();

// 4. To retrieve results returned by the procedure GPXADMIN.QUERY1

ResultSet rs = clstmt.getResultSet();

// Print the results of the XQuery using the result set object.
while(rs.next())
{
    System.out.println(rs.getString(1));
}

```

Insert, update, and delete XML element from XML document.

In this section, you will insert, update, and delete XML elements from an XML document stored in the XML column in a table. The GPX data of CalaSantVicenc will be inserted a second time under ID 21 so you can compare the changes with the original data stored under ID 6.

Use the following file from the included [download](#): InsUpdDelXMLElement.java.

To execute this Java file, execute the following two commands:

```

C:\>javac InsUpdDelXMLElement.java
C:\>java InsUpdDelXMLElement

```

Listing 11. Insert, update, and delete XML element

```

// 1. Follow the steps mentioned in any of the previous steps to load DB2
JDBC Type 4 Driver and establish a connection to DB2.

// 2. Set the current schema to GPXADMIN.

/*
3. Insert an XML Document into an XML column from an external file.
*/

```

```

String insrtStmt = "insert into GPXADMIN.GPX values(?,?,?)";
PreparedStatement pstmt2 = con.prepareStatement(insrtStmt);
pstmt2.setInt(1,21);
pstmt2.setString(2,"CalaSantVicenc_insupddel");
File binFile = new File("./sampledata/CalaSantVicenc.xml");
InputStream inBin = new FileInputStream(binFile);
pstmt2.setBinaryStream(3,inBin,(int)binFile.length());
pstmt2.execute();
inBin.close();

/*
6. INSERT an element in an XML document using the transform expression
in an XQuery statement.
*/

String insEleStmt = "update GPXADMIN.GPX
SET DOCUMENT = XMLQUERY('transform copy $i := $DOCUMENT modify ( do insert
<climate> Hot </climate> after $i/*:gpx/*:wpt/*:ele) return $i') where id=21";
System.out.println("\nXQuery to INSERT an element into a XML document :\n");
System.out.println(insEleStmt);
stmt.executeUpdate(insEleStmt);
System.out.println("\n<climate> element inserted successfully\n");
ResultSet rsInsEle = stmt.executeQuery
("SELECT DOCUMENT FROM GPXADMIN.GPX where ID=21");
while (rsInsEle.next()) {
    SQLXML xmldata = rsInsEle.getSQLXML (1);
    System.out.println (xmldata.getString ());
}

/*
7. UPDATE the value of an existing element in an XML document using the
transform expression in an XQuery statement.
*/

String replaceStmt = "UPDATE GPXADMIN.GPX SET DOCUMENT =
XMLQUERY('transform copy $j := $DOCUMENT modify (
do replace value of $j/*:gpx/*:wpt/*:sym/text() with \"Town\")
return $j') where id = 21";
System.out.println("\nXQuery to UPDATE an element's value from XML document :\n");
System.out.println(replaceStmt);
stmt.executeUpdate(replaceStmt);
System.out.println("\nValue of <sym> element replaced successfully\n");
ResultSet rsRplc = stmt.executeQuery
("SELECT DOCUMENT FROM GPXADMIN.GPX where ID=21");
while (rsRplc.next()) {
    SQLXML xmldata = rsRplc.getSQLXML (1);
    System.out.println (xmldata.getString ());
}

/*
8. DELETE an element from an XML document using the
transform expression in an X statement.
*/

String delStmt2 = "UPDATE GPXADMIN.GPX SET DOCUMENT =
XMLQUERY('transform copy $k := $DOCUMENT modify
(do delete $k/*:gpx/*:wpt/*:ele)
return $k') where id = 21";
System.out.println("\nXQuery to DELETE an element from XML document :\n");
System.out.println(delStmt2);
stmt.executeUpdate(delStmt2);
ResultSet rs = stmt.executeQuery
("SELECT DOCUMENT FROM GPXADMIN.GPX where ID=21");
System.out.println("\nElement <ele> deleted successfully\n");
while (rs.next()) {
    SQLXML xmldata = rs.getSQLXML (1);
    System.out.println (xmldata.getString ());
}

```

```
}
```

Transform XML with XSLTRANSFORM function

Here, you will transform stored XML using the XSLTRANSFORM scalar function.

Use the following file from the included [download](#): XslTransform.java.

To execute this Java file, execute the following two commands:

```
C:\>javac XslTransform.java
```

```
C:\>java XslTransform
```

Listing 12. Transform XML with XSLTRANSFORM function

```
// 1. Follow steps mentioned in any of the above listings to load DB2 JDBC Type 4
// Driver and establish a connection to DB2.

// 2. Create table to store XML data and related XSL stylesheet data
// in two different XML columns.
create table GPXADMIN.TRNSFRM(ID INT, XMLDOC XML, XSLDOC XML)

/*
3. Prepare and execute an insert statement to insert an GPX XML document and an XSL
stylesheet, pass the contents of the XML document and the XSL stylesheet as string
parameters to the con.prepareStatement method.
*/

// 4. Execute the select statement that uses the XSLTRANSFORM scalar function.
String sql2 =
"SELECT XSLTRANSFORM (xmlDOC USING xslDOC AS CLOB (10M)) FROM GPXADMIN.TRNSFRM";
PreparedStatement pstmt2 = con.prepareStatement(sql2);
ResultSet rs2 = pstmt2.executeQuery();
while (rs2.next())
{System.out.println(rs2.getString(1));}
System.out.println("XSLTRANSFORM Function executed successfully");

// 5. Use ResultSet object to retrieve result of the XSLTRANSFORMATION scalar function.
```

Listing 13. Sample output from executing the select statement with XSLTRANSFORM function

```
<html xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xdt="http://www.w3.org/2005/xpath-datatypes"
xmlns:n1="http://www.topografix.com/GPX/1/1"
xmlns:fn="http://www.w3.org/2005/xpath-functions">
<head>
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title></title>
</head>
<body><br>
<h1><span style="font-weight:bold; ">GPX - GPS Data</span></h1>
<h2><span>Waypoints:</span></h2><br><br><br>
<h3><span style="font-style:italic; font-weight:bold; ">
Cala Sant Vicen&aring;x&shy; Mallorca</span></h3>
<span>Latitude&nbsp;: </span>
```

```
<span>39.921055008</span><br>
<span>Longitude: </span>
<span>3.054223107</span><br><br><br>
<span>&nbsp;</span><br>
</body>
</html>
```

Conclusion

This article provided many examples of manipulating and managing XML through Java JDBC 4.0 driver in two categories — administrative tasks and application development tasks. This article covered the use of stored procedures and parameter markers, as well as XML-specific features, such as XQuery, SQL/XML, XSLT and XML schema registration, and XML document validation. With this information, Java developers are able to proceed with creating pureXML applications.

Acknowledgment

With thanks to Tony Avdiu and Manoj Sardana for making improvements to the article and download.

Downloads

Description	Name	Size	Download method
Sample files and Java programs for this article	Work_with_XML_DB2_95_using JDBC	292 KB	HTTP

[Information about download methods](#)

Resources

Learn

- [DB2 Solution Information Center](#): Learn how to use the DB2 family of products and features, as well as related WebSphere® Information Integration products and features. [DB2 JDBC 4.0 support](#) has been added.
- ["DB2 9.1 Developing Java Applications"](#) (IBM, September 2006): Develop Java applications that access DB2 databases, using JDBC and SQLJ.
- ["Using the SQLXML data type"](#) (developerWorks, April 2008): Check out procedures to create an XML document, store an XML document in a relational database, retrieve an XML document from a database, and navigate an XML document with the SQLXML Java data type.
- ["Using DB2 XML and Java"](#) (developerWorks, October 2006): Integrate pureXML into your Java applications, and make development easier with the new DB2 Developer Workbench.
- ["Update XML in DB2 9.5"](#) (developerWorks, October 2007): Get an introduction to the XQuery Update Facility of DB2 9.5. This article presents examples of typical XML update operations and discusses how to avoid common pitfalls.
- [GPX: The GPS Exchange Format](#): Learn more about GPX.
- [GPS: Global Positioning System](#) (Wikipedia): Get more information about GPS.
- [Industry Formats and Services for pureXML](#): Follow a demonstration of end-to-end XML data exchange with a DB2 9 pureXML database.
- [developerWorks Information Management zone](#): Learn more about Information Management. Find technical documentation, how-to articles, education, downloads, product information, and more.
- Stay current with [developerWorks technical events and webcasts](#).
- [Technology bookstore](#): Browse for books on these and other technical topics.

Get products and technologies

- [Industry Formats and Services for pureXML](#): Download the available industry bundles.
- Build your next development project with [IBM trial software](#), available for download directly from developerWorks.

Discuss

- [Participate in the discussion forum for this content](#).
- Participate in [developerWorks blogs](#) and get involved in the developerWorks

community.

About the authors

Khurram Faraaz

Khurram Faraaz is an IBM Certified Database Administrator for DB2 9 for Linux, UNIX, and Windows. He joined IBM in 2006 and has since worked in the DB2 for Linux, UNIX, and Windows XML FVT team. During this time, apart from his daily FVT tasks, he has been involved in developing XML industry bundles and has co-authored another article on XML schema evolution.

Ronny Bartsch

Ronny Bartsch is a software developer and worked at IBM's Information Management Group (part of IBM software Group) in Hawthorne, USA. He developed industry bundles and demos related to industry standards using XML and DB2 pureXML.

Susan Malaika

Susan Malaika works in IBM's Information Management Group. She specializes in XML and Web technologies, including Grid computing. She has published articles and co-edited a book on the Web. She is a member of the IBM Academy of Technology.

Trademarks

IBM, the IBM logo, ibm.com, developerWorks, DB2, pureXML, and WebSphere are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. See the current list of [IBM trademarks](#). Java and all Java-based trademarks and logos are trademarks of Sun Microsystems,

Inc. in the United States, other countries, or both.