

XML application migration from DB2 for z/OS V8 to DB2 9, Part 1: Partial updates to XML documents in DB2 9 for z/OS

The update stored procedure

Skill Level: Intermediate

[John R. Shelton \(johnshel@us.ibm.com\)](mailto:johnshel@us.ibm.com)
Software engineer
IBM

[Hardeep Singh \(hardeep@us.ibm.com\)](mailto:hardeep@us.ibm.com)
Software engineer, DB2 XML Extender
IBM

20 Mar 2008

The first of a three-article series on migrating your XML applications from IBM® DB2® for z/OS® V8 to DB2 9 for z/OS, this article describes a method for performing partial updates to XML documents stored natively in DB2 9, using a stored procedure that's included as a download.

Note: *These three articles were originally written for DB2 for Linux®, UNIX®, and Windows® by Hardeep Singh. They have been modified to be applicable to DB2 for z/OS by John Shelton.*

Introduction

XML support in DB2 for z/OS Version 8 is based on DB2's relational infrastructure. Prior to the DB2 9 release, XML data was either stored intact as a character large object (CLOB) or shredded to relational tables. In contrast, DB2 9 has true native support for XML data. XML is now handled as a new data type, and XML data is stored in a parsed annotated tree that is separate from the relational data storage. XML indexing based on an XML pattern has been introduced, along with support for

SQL/XML as the languages to query and publish XML data. In order to understand the impact of these new XML features on migration, it is necessary to compare the different techniques for storing and querying XML data in the DB2 V8 to similar or new XML functionality available in DB2 9.

This article is the first of a three-article series on migrating XML applications from DB2 V8 to DB2 9. The series starts with a description of a Java™-based stored procedure that you can use for performing sub-document updates for XML data. You can download the source code and a JAR file for the update stored procedure, and follow the instructions for installing it (see [Downloads](#)).

The second article compares the XML features in DB2 V8 with those in DB2 9. It then briefly discusses the new XML features introduced in DB2 9 and goes into the details regarding the impact the new XML support has on migrating existing XML-based applications. This article also has the source code for a Java-based utility to help generate scripts to migrate the database objects.

The last article in the series walks you through a step-by-step sample migration scenario. It includes source code for the sample scenario.

Update stored procedure

For XML documents stored natively in DB2, there is no out-of-the-box functionality for performing sub-document updates. One solution to this problem is to bring the document over to the client, modify it, and then save it back into the database. This approach is limited by the XML capabilities of the client environment and also requires a level of expertise in writing clients based on the document object model (DOM). By creating an update stored procedure it is possible to update XML documents in the database without needing to bring them over to the client. This stored procedure enables partial updates of XML documents stored natively in the database.

The stored procedure allows for:

- Changing the value of any text or attribute nodes in the target XML document
- Replacing an element node (along with all its children) in the XML document with another XML element
- Deleting a node in the XML document
- Inserting a new element
- Multiple updates to the source document
- Updates to multiple source documents

- Replacing another XML document with the modified one
- Inserting the modified document into a new record

The update information can be:

- Statically embedded in the update call
- Dynamically created at runtime using SQL
- Computed using an arithmetic expression on the original text or attribute value

Note: Behind the scene, the update stored procedure still does a full document update.

XMLUPDATE command

DB2XMLFUNCTIONS.XMLUPDATE (commandXML, querySQL, updateSQL, errorCode, errorMsg)

- **commandXML** - This argument is an XML string that encapsulates the update commands. These commands are then applied to the XML documents selected by the querySQL.

The structure of this command is:

```
<updates namespaces="">
<update using="" col="" action="" path="">update value</update>
</updates>
```

- **<updates>** - This is the root element that wraps all the update command elements.
- **@namespace** - The value for this attribute should be `prefix:namespace` strings separated by semicolons. The prefixes are then used in any paths expressions used to navigate in the XML document.
 - **Essential** - No (it is only needed if namespaces are used in any path.).
 - Even default namespaces have to be qualified with a prefix.

- **<update>** - This element defines each modification that needs to be done on the target XML document.
 - **Occurrence** - One or more of these elements can be defined.
 - Each occurrence handles one modification to the document.
- **@col** - The value for this attribute should be a number corresponding to the location of the column being modified in the querySQL.
 - **Essential** - Yes.
 - **Valid value** - Column position starting from one.
- **@path** - The value of this attribute is the XPath location of the node in the target XML document. If the path is invalid, the stored procedure is aborted.
 - **Essential** - Yes.
 - **Valid value** - XPath expression.
 - Be sure to set the namespaces attribute if you are using namespaces in your XPath.
 - You cannot use wildcards for namespaces.
- **@using** - The only valid value for this attribute is **SQL**. If this attribute is present and set to **SQL**, then the `update value` (child node of the `<update>` element) is considered to be an SQL query. The first column from the first row of the query result is used as the new `update value`. If the query fails, the stored procedure is aborted.
 - **Essential** - No.
 - **Valid value** - SQL.
 - For XQuery you can either use the keyword XQuery or embed the XQuery inside your SQL using SQL/XML functions.
- **@action** - This attribute defines the action to be taken on the target node (located using the XPath defined in the `@path` attribute) in the XML document. If the action fails, the stored procedure is aborted.

- **Essential** - No. If action is not set, it is assumed to be a replace.
- **Valid value** - Replace, append, delete, and compute:
 - **replace** - Replace the target node with the `update value`.
 - **append** - Append the `update value` as a child to the target node.
 - **delete** - Delete the target node.
 - **compute** - The `update value` is treated as a parameterized expression. The question marks (?) in the expression are replaced with the existing text value of the target node. The expression is then computed, and the resulting value replaces the existing value in the target node. The XPath for a computed value should be a leaf node only.
- **update value** - This is the child node of each update command (`//update/*`). It can either be a text node or an element.
 - **Essential** - No. You do not need it for `action=delete`.
 - **Valid value** -- When the `@using` attribute is set to `SQL`, the child node should be a text value. It is treated as the SQL expression. When the `@action` attribute is set to `compute`, the child node should be a text value. It is treated as the parameterized expression. In all other cases, the child node is considered as the value to be replaced.
- **querySQL** - Any valid SQL select statement that retrieves the XML document(s) that need to be updated.
 - **Essential** - Yes.
 - **Valid value** - Only XML columns can be selected. If any other column is selected, the stored procedure is aborted.
- **updateSQL** - It represents a parameterized update SQL. The modified XML document is bound to the update SQL as a runtime parameter. It allows a modified XML document to be saved to other XML columns in

the database.

- **Essential** - No. If this argument is null, then updatable cursors are used to modify the selected columns.
- **Important:** When executing the update stored procedure from the command-line processor (CLP), you always need to set the value for the `updateSQL` parameter. If this value is set to null or empty string, CLP throws a JCC exception: *Column not updatable*. It works (`updateSQL` set to null) when you call the update the stored procedure from inside your application code (Java).
- **errorCode** - A value of -1 indicates that the stored procedure was aborted due to some error. If the update was successful, a positive value indicating the number of records that have been updated is returned.
- **errorMsg** - Error messages, including any exception thrown by the XML parser and the JCC driver.

Setting up and removing the stored procedure

Steps to set up the update stored procedure

1. Compile the Java code and create the `db2xmlfunctions.jar` file, by performing the following steps.

Note: The `db2xmlfunctions.jar` can also be downloaded in the [Downloads](#) section. If you opt to download the file, skip to [Step 2](#).

1. Create a directory `/temp/samples`.
 2. Copy the `XMLUpdate_code.zip` (found in the [Downloads](#) section) to your temp directory.
 3. Extract the `XMLUpdate.java` and `XMLParse.java` files to the `/temp/samples` directory.
 4. Compile the Java files and create the JAR file for the UDF.
2. Install the stored procedure in DB2:

```
CALL SQLJ.INSTALL_JAR('file:/u/myhome/samples/db2xmlfunctions.jar' ,
db2xmlfunctions,0);
```

3. Register the stored procedure in your database:

Listing 1. Register the stored procedure

```
CREATE PROCEDURE DB2XML.XMLUPDATE (
    IN    VARCHAR(32000),
    IN    VARCHAR(32000),
    IN    VARCHAR(32000),
    INOUT INTEGER,
    INOUT VARCHAR(32000)
)
DYNAMIC RESULT SETS 0
LANGUAGE JAVA
PARAMETER STYLE JAVA
NO DBINFO
PROGRAM TYPE SUB
COMMIT ON RETURN NO
WLM ENVIRONMENT WLMENVJ
COLLID DSNJDBC
SECURITY DB2
MODIFIES SQL DATA
NOT DETERMINISTIC
EXTERNAL NAME 'db2xmlfunctions:XMLUpdate.update(java.lang.String,java.lang.String,java.lang.Integer,java.sql.ResultSet[])'
STAY RESIDENT NO;
```

Remove the stored procedure

If you make any changes to the stored procedure, you should first uninstall it from DB2 before registering the new version:

```
DROP PROCEDURE DB2XML.XMLUPDATE RESTRICT;
CALL SQLJ.REMOVE_JAR(DB2XMLFUNCTIONS);
```

XMLUpdate samples

For XMLUpdate samples, complete the following steps:

1. Create a test table:

```
Create table XMLCustomer(cid integer not null PRIMARY KEY, info XML );
```

2. Insert a sample XML document into the table:
Listing 2. Insert a sample XML document

```

Insert into XMLCustomer (cid, info ) values (1006 ,
XMLPARSE ( DOCUMENT '
<customerinfo xmlns="
http://posample.org
" Cid="1006">
<name>Hardeep Singh</name>
<addr country="United States">
<street>555 Bailey Ave</street>
<city/>
<prov-state>CA</prov-state>
<pcode-zip> 95141</pcode-zip>
</addr>
<phone type="">543-4610</phone>
</customerinfo>'
PRESERVE WHITESPACE ) );

```

Note: Since the update calls modify the original XML document, you might need to delete the inserted document and reinsert it for some queries.

Example queries

The following are example queries:

1. Replace a node (action=replace).
Update the test document by replacing the simple name element with a complex name element:

Listing 3. Replace a node

```

Call DB2XMLFUNCTIONS.XMLUPDATE (
'<updates namespaces="x:http://posample.org">
<update action="replace" col="1" path="/x:customerinfo/x:name">
<name><fname>Hardeep</fname><lname>Singh</lname></name>
</update>
</updates>',
'Select info from XMLCustomer where cid=1006',
'update XMLCustomer set info=? where cid=1006');

```

2. Update using an SQL query to get the new value (using=SQL):
Listing 4. Update using an SQL query

```

Call DB2XMLFUNCTIONS.XMLUPDATE (
'<updates namespaces="x:http://posample.org">
<update using="sql" action="replace" col="1"
path="//x:customerinfo[@Cid=1006]/x:addr/x:pcode-zip/text()">

```

```
select cid from XMLCustomer where cid=1006
</update>
</updates>',
'Select info from XMLCustomer where cid=1006',
'update XMLCustomer set info=? where cid=1006');
```

3. Compute the value with the given expression (action=compute): **Listing 5. Compute value with expression**

```
Call DB2XMLFUNCTIONS.XMLUPDATE (
'<updates namespaces="x:http://posample.org">
<update action="compute" col="1"
path="/x:customerinfo/x:addr/x:pcode-zip/text()">
(20+?)*32-?
</update>
</updates>',
'Select info from XMLCustomer where cid=1006',
'update XMLCustomer set info=? where cid=1006');
```

4. Multiple actions on the target XML document: **Listing 6. Multiple actions on target XML document**

```
Call DB2XMLFUNCTIONS.XMLUPDATE (
'<updates namespaces="x:http://posample.org">
<update using="sql" action="replace" col="1"
path="/x:customerinfo/x:addr/x:pcode-zip/text()">
select cid from XMLCustomer where cid=1006
</update>
<update action="compute" col="1"
path="/x:customerinfo/x:addr/x:pcode-zip/text()">
(2+?)*10-?
</update>
<update action="delete" col="1" path="/x:customerinfo/x:name"/>
</updates>',
'Select info from XMLCustomer where cid=1006',
'update XMLCustomer set info=? where cid=1006');
```

5. Validate the document when you update it. For this, you need to create and register the schema in XSR. **Listing 7. Validate document**

```
Call DB2XMLFUNCTIONS.XMLUPDATE (
'<updates namespaces="x:http://posample.org">
<update using="sql" action="replace" col="1"
path="/x:customerinfo/x:addr/x:pcode-zip/text()">
select cid from XMLCustomer where cid=1006
</update>
</updates>',
'Select info from XMLCustomer where cid=1006',
```

```
'update XMLCustomer set info=XMLPARSE(DOCUMENT SYSFUN.DSN_XMLVALIDATE(cast(?
as clob),'SYSXSR.testschema2')) where cid=1006')
```

6. Use XMLUpdate to replace an attribute value: Listing 8. Replace an attribute value

```
Call DB2XMLFUNCTIONS.XMLUPDATE (
'<updates namespaces="x:http://posample.org">
<update action="replace" col="1"
path="/x:customerinfo/x:phone/@type">
tie line
</update>
</updates>',
'Select info from XMLCustomer where cid=1006',
'update XMLCustomer set info=? where cid=1006');
```

7. Use XMLUpdate to replace a text value: Listing 9. Replace a text value

```
Call DB2XMLFUNCTIONS.XMLUPDATE (
'<updates namespaces="x:http://posample.org">
<update action="replace" col="1"
path="/x:customerinfo/x:addr/x:city/text()">
San Jose
</update>
</updates>',
'Select info from XMLCustomer where cid=1006',
'update XMLCustomer set info=? where cid=1006');
```

Important: It is necessary to specify text() at the end of the path. This step ensures that even an empty element (one with no existing text node) is updated. If text() is omitted and there is no existing text value to replace, the update command fails.

8. Use XMLUpdate to append a child node: Listing 10. Append a child node

```
Call DB2XMLFUNCTIONS.XMLUPDATE (
'<updates namespaces="x:http://posample.org">
<update action="append" col="1" path="/x:customerinfo/x:addr">
<county>Santa Clara</county>
</update>
</updates>',
'Select info from XMLCustomer where cid=1006',
'update XMLCustomer set info=? where cid=1006');
```

Note: The new node <county> is not in any namespace.

9. Use XMLUpdate to insert the updated XML to a new row:
Listing 11. Insert the updated XML to a new row

```
Call DB2XMLFUNCTIONS.XMLUPDATE (
'<updates namespaces="x:http://posample.org">
<update action="replace" col="1"
path="/x:customerinfo/x:name">
<name>Marja Soinen</name>
</update>
<update action="replace" col="1"
path="/x:customerinfo/@Cid">1008</update>
</updates>',
'Select info from XMLCustomer where cid=1006',
'insert into XMLCustomer (cid, info ) values (1008, cast( ? as xml))');
```

10. Use XMLUpdate to delete a node:
Listing 12. Delete a node

```
Call DB2XMLFUNCTIONS.XMLUPDATE (
'<updates namespaces="x:http://posample.org">
<update action="delete" col="1" path="/x:customerinfo/x:name"/>
</updates>',
'Select info from XMLCustomer where cid=1006',
'update XMLCustomer set info=? where cid=1006');
```

11. When @action is not set in the update element, it defaults to replace:
Listing 13. Update defaults to replace

```
Call DB2XMLFUNCTIONS.XMLUPDATE (
'<updates namespaces="x:http://posample.org">
<update col="1" path="//x:customerinfo[@Cid=1006]/x:phone">
<phone><areacode>910</areacode></phone>
</update>
</updates>',
'Select info from XMLCustomer where cid=1006',
'update XMLCustomer set info=? where cid=1006');
```

12. The following example shows XMLUpdate with invalid namespace or a namespace without a prefix:
Listing 14. Invalid namespace or a namespace without a prefix

```
Call DB2XMLFUNCTIONS.XMLUPDATE (
'<updates namespaces="x:http://my.org">
<update col="1"
path="//x:customerinfo[@Cid=1006]/x:phone">
<phone><areacode>910</areacode></phone>
</update>
</updates>',
```

```
'Select info from XMLCustomer where cid=1006',
'update XMLCustomer set info=? where cid=1006');
```

This query returns with the error set to 1 and the following error message:
 <error type='abort' action='replace' msg='Cannot find path //x:customerinfo[@Cid=1006]/x:phone) in the XMLDocument '>

13. The following example shows XMLUpdate with a missing path in the update element:

Listing 15. Missing path in the update element

```
Call DB2XMLFUNCTIONS.XMLUPDATE (
'<updates > <update col="1"> (20+?)*32-? </update></updates>',
'Select info from XMLCustomer where cid=1006',
'update XMLCustomer set info=? where cid=1006');
```

This query returns with the error set to 1 and the following error message:
 <error type='abort' action='null' msg='path not defined'></error>

14. Replace a node with a new node in the same namespace:

Listing 16. Replace a node with a new node

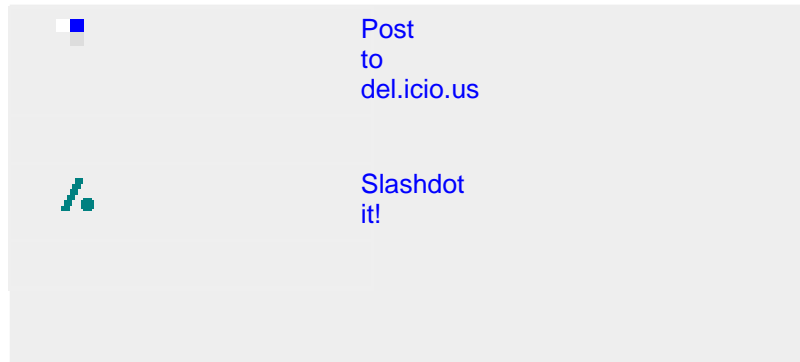
```
'<updates namespaces="x:http://posample.org">
<update action="replace" col="1" path="/x:customerinfo/x:name">
<name xmlns="http://posample.org">
<fname>Marja</fname><lname>Soininen</lname>
</name>
</update>
</updates>',
'Select info from XMLCustomer where cid=1008',
'insert into XMLCustomer (cid, info ) values (1007, cast( ? as xml))');
```

Conclusion

Share this...



Digg
this
story



The `update` stored procedure described in this article enables partial updates to XML documents stored natively in the database. The next article delves deeper and explores, in detail, the impact the new XML support has on migrating existing XML-based applications.

Acknowledgements

Thanks to Jason Cu and Manogari Simanjuntak for their help with converting the original article to z/OS.

Downloads

Description	Name	Size	Download method
Update stored procedure classes	db2xmlfunctions.zip	13KB	HTTP
Update stored procedure source code	XMLUpdate_code.zip	9KB	HTTP

[Information about download methods](#)

Resources

Learn

- [developerWorks DB2 for z/OS page](#): Read articles and tutorials and connect to other resources to expand your DB2 skills.
- ["XML application migration from DB2 for Linux, UNIX, and Windows 8.2 to DB2 9, Part 1"](#) (developerWorks, May 2006): Read the original article for DB2 for Linux, UNIX, and Windows.
- [developerWorks Information Management zone](#): Learn more about Information Management. Find technical documentation, how-to articles, education, downloads, product information, and more.
- [Technology bookstore](#): Browse for books on these and other technical topics.
- Stay current with [developerWorks technical events and webcasts](#).

Get products and technologies

- Download [IBM product evaluation versions](#) and get your hands on application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.

Discuss

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

About the authors

John R. Shelton

John Shelton is a member of the DB2 XML Extender team.

Hardeep Singh

Hardeep Singh is a member of the DB2 Native XML team. He is the architect for DB2 XML tooling. He has more than 21 years of industry experience.