Education: Papers

Building an XML Application, Step 2: Generating XML from a Data Store

Doug Tidwell
IBM XML Technical Strategy Group, TaskGuide Development
Updated January 1999

Step 1: Writing a DTD
Step 3: Converting XML into HTML with the Document Object Model (DOM)

Abstract: In this paper, we'll discuss how to generate XML-tagged data from a data store. In our examples here, we'll be using IBM DB2 for Windows NT as our data store. Unlike the previous paper, in which we produced a DTD, in this paper we'll focus on building Java code. Although you could certainly build this application with other languages, we'll use Java here for portability's sake. We'll also discuss all of the parts of our Java code, even though there are tools (most notably IBM VisualAge for Java and IBM WebSphere Studio) that could generate this code for us.

Technologies We'll be Using

In addition to XML, other technologies we'll use here are the application programming interfaces (APIs) for the Java Database Connection (JDBC) and Java Servlets. JDBC has been implemented by a wide range of database vendors, including IBM. Using JDBC makes it easy to modify our sample application to use some other JDBC-compliant database server. The Servlet interface allows us to build Java code that runs on the server and delivers its output (typically HTML) to the client.

XML and Database Structures

If you need to generate XML from a query of some sort, it's best that you have a particular DTD in mind. In this paper, we'll continue working with the DTD we generated earlier. To refresh your memory, here's our final DTD:

```
<!-- flights.dtd -->
<!ELEMENT flights (itinerary)+>

<!ELEMENT itinerary (outbound-depart-from, outbound-depart-time,
                     outbound-arrive-in, outbound-arrive-time,
                     outbound-airline, returning-depart-from,
                     returning-depart-time, returning-arrive-in,
                     returning-arrive-time, returning-airline)>

<!ELEMENT outbound-depart-from     (#PCDATA)>
<!ELEMENT outbound-depart-time     (EMPTY)>
<!ATTLIST outbound-depart-time     year      CDATA #REQUIRED
                                   month     CDATA #REQUIRED
                                   day       CDATA #REQUIRED
                                   hour      CDATA #REQUIRED
                                   minute    CDATA #REQUIRED>
```

```
<!ELEMENT outbound-arrive-in       (#PCDATA)>
<!ELEMENT outbound-arrive-time     (EMPTY)>
<!ATTLIST outbound-arrive-time     year       CDATA #REQUIRED
                                   month      CDATA #REQUIRED
                                   day        CDATA #REQUIRED
                                   hour       CDATA #REQUIRED
                                   minute     CDATA #REQUIRED>
<!ELEMENT outbound-airline         (EMPTY)>
<!ATTLIST outbound-airline         flightNum CDATA #REQUIRED
                                   carrierName (Alitalia | American | Delta |
                                               Northwest | Pacific |
                                               TWA | United) "American">
<!ELEMENT returning-depart-from    (#PCDATA)>
<!ELEMENT returning-depart-time    (EMPTY)>
<!ATTLIST returning-depart-time    year       CDATA #REQUIRED
                                   month      CDATA #REQUIRED
                                   day        CDATA #REQUIRED
                                   hour       CDATA #REQUIRED
                                   minute     CDATA #REQUIRED>
<!ELEMENT returning-arrive-in      (#PCDATA)>
<!ELEMENT returning-arrive-time    (EMPTY)>
<!ATTLIST returning-arrive-time    year       CDATA #REQUIRED
                                   month      CDATA #REQUIRED
                                   day        CDATA #REQUIRED
                                   hour       CDATA #REQUIRED
                                   minute     CDATA #REQUIRED>
<!ELEMENT returning-airline        (EMPTY)>
<!ATTLIST returning-airline        flightNum CDATA #REQUIRED
                                   carrierName (Alitalia | American | Delta |
                                               Northwest | Pacific |
                                               TWA | United) "American">

<!ENTITY xt "Xtreme Travel">
```

Given this DTD, we need to take another look at the database structure. The data we receive from DB2 will be records containing fields according to the following structure:

Structure of Database Table **flights**			
Column Name	**Sample Data**	**Column Name**	**Sample Data**
id	0001	DepartFrom_1	Chicago
DepartFrom_2	Palm Springs	DepartTime_1	January 10 1999 6:30 AM
DepartTime_2	January 15 1999 11:50 AM	ArriveIn_1	Palm Springs
ArriveIn_2	Chicago	ArriveTime_1	January 10 1999 11:03 AM
ArriveTime_2	January 15 1999 9:24 PM	Airline_1	American #303
Airline_2	American #1250		
Note: As we mentioned in the previous paper, the id field is ignored.			

Let's Get Started!

Now that we've reviewed the basics, our task is to convert the native database format into an XML document. Let's look at the Java code that retrieves the data from DB2.

Initialize the JDBC Driver

Before we can interact with DB2, we must initialize its JDBC driver program. The driver takes JDBC instructions as input, then converts them into instructions that DB2 understands.

```
static
{
  try
  {
    Class.forName("COM.ibm.db2.jdbc.app.DB2Driver").newInstance();
  }
  catch (Exception e)
  {
    System.out.println("Can't get the driver!");
    e.printStackTrace();
  }
}
```

This code is invoked the first time the servlet is loaded. Once the driver is loaded, we can begin to interact with the database. Note that if you're using a database other than DB2, the class name above (COM.ibm.db2.jdbc.app.DB2Driver) would be different.

Connect to the Database

Our next step is to connect to the database. This is done by creating a JDBC Connection object.

```
try
{
  con = DriverManager.getConnection("jdbc:db2:wbsphere");
}
catch (Exception e)
{
  e.printStackTrace();
}
```

The code above creates a connection to the database using the JDBC Universal Resource Locator (URL) format. In the example above, all URLs begin with jdbc; the second component of the URL is defined by the database vendor (DB2 uses the refreshingly intuitive string db2), and the third component is the name of the database from which we'll be extracting data (wbsphere).

Executing Structured Query Language (SQL) Statements

Now that we're connected to the database, we need to create an SQL statement and execute it. This will return a JDBC ResultSet object; the ResultSet object contains all the data returned by the query. Here's the code that executes the SQL statement:

```
String query = "select * from flights";
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery(query);
```

In this case, we're returning all the columns and all the rows from the table flights in the database wbsphere. A real world example would most likely get user input to modify the query, and would probably retrieve no more data than necessary.

```
// Another sample query, more realistic than the one we're using
String query = "select departtime_1, arrivein_1, airline_1 " +
  "from wbsphere where departfrom_1 = 'Chicago'";
```

In the example above, the query limits the data returned by asking for certain database columns only, and by returning only data from records in which the value of departfrom_1 is Chicago.

Converting a ResultSet into XML

At this point, we've connected to the database and executed a query. The ResultSet object returned by the JDBC library contains the results of the query. Before we begin to examine the result set, we'll write out the XML header:

```
out.println("<?xml version=\"1.0\" ?>\n");
out.println("<!DOCTYPE travelplans SYSTEM \"flights.dtd\">\n\n");
```

Now we begin our XML data stream. From our DTD, we know we have to begin with a <travelplans> tag:

```
out.println("<travelplans>\n");
```

Before we begin to convert the results of our database query into an XML document, we'll set up some Java objects to correctly process the dates and times stored in the database. This information comes to us as text; we can use Java's national language functions to store date and time information in a language-independent way. We'll create a Java Calendar object, as well as two DateFormat objects to handle the date and time functions for us.

```
Calendar cal = Calendar.getInstance();
String currentYear = Integer.toString(cal.get(Calendar.YEAR));
DateFormat df1 = DateFormat.getDateInstance(DateFormat.LONG);
DateFormat df2 = DateFormat.getTimeInstance(DateFormat.SHORT);
```

Now that we've begun our XML document, we'll look at each record in the ResultSet, and generate the appropriate XML tags from the data. According to our DTD, each record should be wrapped in an <itinerary> tag. Each record in the ResultSet contains 11 strings that correspond to the 11 fields in the flights table. With the exception of the id field, we'll generate an XML tag from the strings we want. We use the rs.getString() method to do this:

```
while (rs.next())
{
  out.print("  <itinerary>\n" +
            "    <outbound-depart-from>" + rs.getString(2).trim() +
            "</outbound-depart-from>\n");

  // All of the time fields have to be processed this way.  We use Java's
  // DateFormat and Calendar classes to parse the data from the database.
  // Note that we have to parse the date and time separately because of the way they're
  // stored in the database.
  try
  {
    st = new StringTokenizer(rs.getString(3).trim());
    String dayPortion = st.nextToken() + " " + st.nextToken() + ", " +
                        currentYear;
    cal.setTime(df1.parse(dayPortion));
  }
  catch (ParseException pe)
  {
    cal = Calendar.getInstance();
  }

  out.print("    <outbound-depart-time year=\"" + currentYear + "\" ");
  out.print("month=\"" + cal.get(Calendar.MONTH) + "\" ");
  out.print("day=\"" + cal.get(Calendar.DAY_OF_MONTH) + "\" ");

  try
  {
    st = new StringTokenizer(rs.getString(3).trim());
    st.nextToken();
    st.nextToken();
    String timePortion = st.nextToken() + " " + st.nextToken();
    cal.setTime(df2.parse(timePortion));
  }
  catch (ParseException pe)
  {
    cal = Calendar.getInstance();
  }
  out.print("hour=\"" + cal.get(Calendar.HOUR_OF_DAY) + "\" ");
  out.print("minute=\"" + cal.get(Calendar.MINUTE) + "\" />\n");
  // End of tedious date parsing code

  out.print("    <outbound-arrive-in>" + rs.getString(4).trim() +
            "</outbound-arrive-in>\n");

  try
  {
```

```
    cal.setTime(df.parse(rs.getString(5).trim()));
  }
  catch (ParseException pe)
  {
    cal = Calendar.getInstance();
  }

  out.print("    <outbound-arrive-time year=\"1999\" ");
  out.print("month=\"" + cal.get(Calendar.MONTH) + "\" ");
  out.print("day=\"" + cal.get(Calendar.DAY_OF_MONTH) + "\" ");
  out.print("hour=\"" + cal.get(Calendar.HOUR_OF_DAY) + "\" ");
  out.print("minute=\"" + cal.get(Calendar.MINUTE) + "\" />\n");

  // We have to parse field 6 to convert it into attributes
  StringTokenizer st = new StringTokenizer(rs.getString(6).trim());
  out.print("  <outbound-airline carrierName=\"");
  String start = new String();
  String end = new String();
  while (st.hasMoreTokens())
  {
    start += " " + end;
    end = st.nextToken();
  }
  out.print(start + "\" flightNum = \" +
            end + "\" />\n");
  out.print("    <returning-depart-from>" + rs.getString(7).trim() +
          "</returning-depart-from>\n");

  // The <returning-depart-time> tag is processed as above

  out.print("    <returning-arrive-in>" + rs.getString(9).trim() +
          "</returning-arrive-in>\n");

  // The <returning-arrive-time> tag is processed as above

  // We have to parse field 11 to convert it into attributes
  StringTokenizer st = new StringTokenizer(rs.getString(11).trim());
  out.print("  <returning-airline carrierName=\"");
  String start = new String();
  String end = new String();
  while (st.hasMoreTokens())
  {
    start += " " + end;
    end = st.nextToken();
  }
  out.print(start + "\" flightNum = \" +
            end + "\" />\n");
  out.print("</itinerary>\n");
}
```

This while loop processes each record in the ResultSet. Notice that we started with the second string; rs.getString(1) returns the id field we've chosen to ignore. Also notice that we had to do some extra work to convert the database fields Airline_1 and Airline_2 into attributes of the <outbound-airline> and <returning-airline> tags.

Once the data has been processed, we'll close our document and clean up the database connection.

```
out.print("</travelplans>\n");

rs.close();
stmt.close();
con.close();
```

Sample XML Document

Here's a look at part of an XML document generated by this servlet:

```
<?xml version="1.0" ?>
<!DOCTYPE travelplans SYSTEM "flights.dtd">
  <travelplans>
    <itinerary>
      <outbound-depart-from>Chicago</outbound-depart-from>
      <outbound-depart-time year="1999" month="1" day="10" hour="6" minute="30" />
      <outbound-arrive-in>Palm Springs</outbound-arrive-in>
      <outbound-arrive-time year="1999" month="1" day="10" hour="11" minute="3" />
```

```
    <outbound-airline carrierName="American" flightNum="303" />
    <returning-depart-from>Palm Springs</returning-depart-from>
    <returning-depart-time year="1999" month="1" day="15" hour="11" minute="50" />
    <returning-arrive-in>Chicago</returning-arrive-in>
    <returning-arrive-time year="1999" month="1" day="15" hour="21" minute="24" />
    <returning-airline carrierName="American" flightNum="1250" />
  </itinerary>
  <itinerary>
    <outbound-depart-from>Atlanta</outbound-depart-from>
    <outbound-depart-time year="1999" month="1" day="10" hour="7" minute="0" />
    <outbound-arrive-in>Palm Springs</outbound-arrive-in>
    <outbound-arrive-time year="1999" month="1" day="10" hour="10" minute="12" />
    <outbound-airline carrierName="Delta" flightNum="1421" />
    <returning-depart-from>Palm Springs</returning-depart-from>
    <returning-depart-time year="1999" month="1" day="15" hour="16" minute="0" />
    <returning-arrive-in>Atlanta</returning-arrive-in>
    <returning-arrive-time year="1999" month="1" day="15" hour="22" minute="38" />
    <returning-airline carrierName="Delta" flightNum="5906" />
  </itinerary>
  . . .
</travelplans>
```

Sample Code

To study this code, see the html version of this file on the XML web site.

flights.dtd

The DTD for our sample data

generateXML.java

Java source file for the servlet that generates XML

createdb.bat

An MS-DOS batch file that creates the DB2 database

flights.txt

File of comma-separated values that represent the sample database.

Summary

In this paper, we learned how to retrieve information from a data store, then format that data as an XML document. We used our previously-created DTD to format the data.

What's Next?

Our next paper is building a Document Object Model (DOM) tree from this XML document. Building the DOM tree is the first step towards transforming our XML data into some other format (such as HTML) that can be consumed by a target application.

Step 1: Writing a DTD
Step 3: Converting XML into HTML with the Document Object Model (DOM)

Please send any comments or questions to:
Doug Tidwell
dtidwell@us.ibm.com