IBM

developerWorks.

Optional XML in relational databases, Part 2: Create, store, and manipulate optional XML data with JAXB and Java annotations

Move XML-defined optional data into a relational database

Skill Level: Intermediate

Stephen B Morris
CTO
Omey Communications

07 Jul 2009

Explore the software required to create, store, and manipulate optional XML relational data in this article—the second in a two-part series. The software used includes fully worked code examples with Java™ Architecture for XML Binding (JAXB), the Java Persistence API (JPA)/Hibernate, an in-memory database, and persistence-related annotations.

Other articles in this series

- Optional XML in relational databases, Part 1: Are null values needed?

In Part 1 of this two-part series, you saw how to use JAXB to transform from the XML domain into Java code. This notion of inter-domain transformation is useful in many fields of study—for example, moving from the time domain into the frequency domain in the field of signal processing. A related example is that of XSLT, where a stylesheet is used to transform XML into some other text format, such as HTML. Now, look at the setup required for JAXB.

Setting up JAXB

© Copyright IBM Corporation 2009. All rights reserved.

Usually, I don't say much about software installation, but for the Java Web Services Developer Pack (see Resources), installation is a bit trickier than usual. So, I describe the process to help you get everything up and running.

To begin, download version 2.0 of the pack. The Web services download is relatively small—23MB—and shouldn't take too long. If you want to read more after this article, you can also download the Web services documentation pack (available on the same page). To run the InstallShield wizard, double-click the downloaded executable file, then follow the instructions. I opted not to install a Web container. Also, for my installation directory, I used C:\Sun\jwsdp-2.0.

Frequently used acronyms

- API: Application programming interface
- HTML: Hypertext Markup Language
- SQL: Structured Query Language
- XML: Extensible Markup Language
- XSD: XML Schema Definition
- XSLT: Extensible Stylesheet Language Transformation

You've just one really important thing to do after you install the download: Create a folder called *endorsed* inside the %JAVA_HOME%\jre\lib directory. Then, paste the contents of the C:\Sun\jwsdp-2.0\lib folder into the new folder, %JAVA_HOME%\jre\lib\endorsed. The %JAVA_HOME% environment variable is the full path to your Java software development kit (JDK) installation. You can, if you prefer, use the `java.endorsed.dirs` system property on the command line, but the new directory creation is a slightly simpler mechanism.

One other item you need is a copy of Apache Ant (see Resources). Download and install Ant, and make sure the executable program is on your system PATH.

Running the source code

The source code for this article is included as a .zip file in Download. To follow along with the examples in this article, just extract the file's contents into a folder such as C:\article_code. You should see two folders beneath this directory: one called unmarshal-read and the another called dbcode.

Running one of the JAXB examples

To make sure your environment is correctly configured, open a DOS console in one of the example folders (for example, C:\Sun\jwsdp-2.0\jaxb\samples\unmarshal-read). Try to run an Ant target in this

© Copyright IBM Corporation 2009. All rights reserved.

folder, as illustrated in Listing 1.

Listing 1. Running Ant

```
C:\Sun\jwsdp-2.0\jaxb\samples\unmarshal-read>ant -p
Buildfile: build.xml
This sample application demonstrates how to unmarshal
an instance document into a Java content tree and access
data contained within it.

Main targets:
clean    Deletes all the generated artifacts.
compile  Compile all Java source files
javadoc  Generates javadoc
run      Run the sample app
Default target: run
```

Once you see program output like Listing 1, you know that your setup is okay. In passing, note that the `ant -p` command is handy to determine the targets supported by a given build.xml file. It gives you the option of peeking inside a build script without running any of the targets.

To run the example program, just type the `ant` command with no parameters. You should see the program output in Listing 2.

Listing 2. Running a JAXB example program

```
C:\Sun\jwsdp-2.0\jaxb\samples\unmarshal-read>ant
Buildfile: build.xml
compile:
[echo] Compiling the schema...
[xjc] C:\Sun\jwsdp-2.0\jaxb\samples\unmarshal-read\gen-src\primer.po is not

found and thus excluded from the dependency check
[xjc] Compiling file:/C:/Sun/jwsdp-2.0/jaxb/samples/unmarshal-read/po.xsd
[xjc] Writing output to C:\Sun\jwsdp-2.0\jaxb\samples\unmarshal-read\gen-src
[echo] Compiling the java source files...
[javac] Compiling 4 source files to
C:\Sun\jwsdp-2.0\jaxb\samples\unmarshal-read\classes

run:
[echo] Running the sample application...
[java] Ship the following items to:
[java]     Alice Smith
[java]     123 Maple Street
[java]     Cambridge, MA 12345
[java]     US
[java]
[java]     5 copies of "Nosferatu - Special Edition (1929)"
[java]     3 copies of
"The Mummy (1959)"
[java]     3 copies of "Godzilla and Mothra:
Battle for Earth/Godzilla vs. King Ghidora"
BUILD SUCCESSFUL
Total time: 9 seconds
```

Listing 2 illustrates a full JAXB program. You'll soon see another example of this

© Copyright IBM Corporation 2009. All rights reserved.

technology for transforming XML into Java code. (A note on the terminology is important here. One speaks of *unmarshalling* XML into Java code: You transform Java code into XML when you *marshall* and XML to Java code when you *unmarshal.*) Now, look at the XML data that contains an optional element.

The XSD specification with an optional element

Listing 3 illustrates the `xsd:complexType` element, which is of interest to you.

Listing 3. An XSD type definition with an optional element

```
<xsd:complexType name="PurchaseOrderType">
<xsd:sequence>
<xsd:element name="shipTo" type="EuropeanAddress"/>
<xsd:element name="billTo" type="EuropeanAddress"/>
<xsd:element ref="comment" minOccurs="0"/>
<xsd:element name="items" type="Items"/>
</xsd:sequence>
<xsd:attribute name="orderDate" type="xsd:date"/>
</xsd:complexType>
```

The contents of Listing 3 are part of a file called *po.xsd,* which is a schema for the XML you'll look at. Listing 4 shows an XML file (called *po.xml*) that conforms to the po.xsd schema.

Listing 4. An example XML document

```
<?xml version="1.0"?>
<purchaseOrder orderDate="1999-10-20">
    <shipTo country="IE">
        <name>Alice Smith</name>
        <street>123 Maple Street</street>
        <city>Cambridge</city>
        <postcode>12345>/postcode>
    </shipTo>
    <billTo country="IE">
        <name>Robert Smith</name>
        <street>8 Oak Avenue</street>
        <city>Cambridge</city>
        <postcode>12345</postcode>
    </billTo>
    <items>
      <item partNum="242-NO" >
         <productName>Nosferatu - Special Edition
(1929)>/productName>
         <quantity>5</quantity>
         <USPrice>19.99</USPrice>
       </item>
      <item partNum="242-MU" >
         <productName>The Mummy (1959)</productName>
         <quantity>3</quantity>
         <USPrice>19.98</USPrice>
       </item>
      <item partNum="242-GZ" >
         <productName>Godzilla and Mothra: Battle for
Earth/Godzilla vs. King Ghidora</productName>
```


© Copyright IBM Corporation 2009. All rights reserved.

```
          <quantity>3</quantity>
          <USPrice>27.95</USPrice>
        </item>
    <>/items>
</purchaseOrder>
```

Notice that the XML in Listing 4 does not contain any comment field. As seen in Listing 3, the comment field is the optional.

You now want to unmarshal the XML data in Listing 4 into one or more Java classes. To do this, open a DOS console in the source code folder unmarshal-read. Then, run the following command:

```
ant compile -Djwsdp.home=C:\Sun\jwsdp-2.0
```

Listing 5 illustrates the expected output.

Listing 5. XML-to-Java transformation

```
C:\article_code\unmarshal-read>ant compile -
Djwsdp.home=C:\Sun\jwsdp-2.0
Buildfile: build.xml

compile:
     [echo] Compiling the schema...
    [mkdir] Created dir: C:\article_code\unmarshal-
read\gen-src
      [xjc] C:\article_code\unmarshal-read\gen-
src\primer.po is not found and thus excluded from the
dependency check
      [xjc] Compiling file:/C:/article_code/unmarshal-
read/po.xsd
      [xjc] Writing output to C:\article_code\unmarshal-
read\gen-src
     [echo] Compiling the java source files...
    [mkdir] Created dir: C:\article_code\unmarshal-
read\classes
    [javac] Compiling 5 source files to
C:\article_code\unmarshal-read\classes

BUILD SUCCESSFUL
Total time: 5 seconds
```

Listing 5 illustrates the magic of JAXB. It generates Java classes from the XML files and is driven by the Java program illustrated in Listing 6.

Listing 6. The code that drives the unmarshalling process

```
public static void main( String[] args ) {
        try {
            // create a JAXBContext capable of handling classes generated into
            // the primer.po package
            JAXBContext jc = JAXBContext.newInstance( "primer.po" );
```


© Copyright IBM Corporation 2009. All rights reserved.

```
            // create an Unmarshaller
            Unmarshaller u = jc.createUnmarshaller();

            // unmarshal a po instance document into a tree of Java content
            // objects composed of classes from the primer.po package.
            JAXBElement<?> poElement =
      (JAXBElement<?>)u.unmarshal( new FileInputStream( "po.xml" ) );
            PurchaseOrderType po = (PurchaseOrderType)poElement.getValue();

            // examine some of the content in the PurchaseOrder
            System.out.println( "Ship the following items to: " );

            // display the shipping address
            EuropeanAddress address = po.getShipTo();
            displayAddress( address );

            // display the items
            Items items = po.getItems();
            displayItems( items );

        } catch( JAXBException je ) {
            je.printStackTrace();
        } catch( IOException ioe ) {
            ioe.printStackTrace();
        }
    }
```

The code in Listing 6 consists of the following main steps:

1. Create a JAXB context.
2. Use the context to create an unmarshaller.
3. Unmarshal the file po.xml.
4. Create an object of the `PurchaseOrderType` class.
5. Extract and display some of the elements from the object.

The Listing 6 code packs quite a lot of power. You now have a Java representation of the original XML data. Given this, you now want to push this data into a relational database.

Transformation from Java code into database entities

The power of Java annotations really comes into its own when you want to persist Java classes. This mapping from Java code into the relational domain is called *object relational mapping* (ORM). Listing 7 provides an example mapping of the `ShippingAddress` class.

Listing 7. An ORM for ShippingAddress

© Copyright IBM Corporation 2009. All rights reserved.

```
@Embeddable
public class ShippingAddress {

    @Column(name = "SHIPPING_ADDRESS_STREET")
    private String street;

    @Column(name = "SHIPPING_ADDRESS_CITY")
    private String city;

    ShippingAddress() {}

    public ShippingAddress(String street, String city) {
        this.street = street;
        this.city = city;
    }

    public String getStreet() {
        return street;
    }

    private void setStreet(String street) {
        this.street = street;
    }

    public String getCity() {
        return city;
    }

    private void setCity(String city) {
        this.city = city;
    }
}
```

If you look at Listing 7 and for the moment forget about the annotations, it's just a simple Java class called `ShippingAddress`. In fact, using an integrated development environment (IDE) such as Eclipse, you can easily auto-generate much of the Listing 7 code—that is, the constructors and the setter and getter methods. The annotations are simply used to map the class and its properties into the relational domain. The `@Embeddable` annotation indicates that this entire class can be embedded into another class (called an *entity class*), as in Listing 8.

Listing 8. The entity class

```
@Entity
@Table(name = "PURCHASE_ORDERS")
public class PurchaseOrder {

    @Id @GeneratedValue
    @Column(name = "PO_ID")
    private Long id;

    @Embedded
    private ShippingAddress shippingAddress;
    @Embedded
    private BillingAddress billingAddress;

    @Column(name = "COMMENT")
    private String comment;
    @Column(nullable=false, name="COMMENT_ENTERED",
columnDefinition="boolean default false")
    private boolean commentEntered;
```


© Copyright IBM Corporation 2009. All rights reserved.

```
// More methods here
}
```

Notice in Listing 8 the way the `ShippingAddress` class instance was embedded into the `PurchaseOrder` class. How does this entity class tie in with a database?

Running the database code

To run the database code, open a DOS console in the folder called dbcode. Create two additional DOS consoles by typing the `start` command. In one of the DOS consoles, run the Ant target `ant startdb`.

This target starts the in-memory HyperSQL DataBase (HSQLDB). After this command you should see output similar to that in Listing 9.

Listing 9. Running HSQLDB

```
C:\article_code\dbcode>ant startdb
Buildfile: build.xml

startdb:
     [java] [Server@1e0be38]: [Thread[main,5,main]]: checkRunning(false)
entered
     [java] [Server@1e0be38]: [Thread[main,5,main]]: checkRunning(false) exited
     [java] [Server@1e0be38]: Startup sequence initiated from main() method
     [java] [Server@1e0be38]: Loaded properties from
[C:\article_code\dbcode\server.properties]
     [java] [Server@1e0be38]: Initiating startup sequence...
     [java] [Server@1e0be38]: Server socket opened successfully in 62 ms.
     [java] [Server@1e0be38]: Database [index=0, id=0, db=file:database/db,
alias=] opened successfully in 313 ms.
     [java] [Server@1e0be38]: Startup sequence completed in 375 ms.
     [java] [Server@1e0be38]: 2009-04-10 09:54:26.625 HSQLDB server 1.8.0 is
online
     [java] [Server@1e0be38]: To close normally, connect and execute SHUTDOWN
SQL
     [java] [Server@1e0be38]: From command line, use [Ctrl]+[C] to abort
abruptly
```

After running the Ant target in Listing 9, you have a running database manager with (as yet) no database. The next task is to run the `schemaexport` Ant target, which creates the database schema in Listing 10.

Listing 10. Database creation

```
C:\article_code\dbcode>ant schemaexport
Buildfile: build.xml

compile:
    [mkdir] Created dir: C:\article_code\dbcode\build
    [javac] Compiling 4 source files to C:\article_code\dbcode\build

copymetafiles:
```


© Copyright IBM Corporation 2009. All rights reserved.

```
     [copy] Copying 2 files to C:\article_code\dbcode\build

schemaexport:
[hibernatetool] Executing Hibernate Tool with a JPA Configuration
[hibernatetool] 1. task: hbm2ddl (Generates database schema)
[hibernatetool]
[hibernatetool]     drop table PURCHASE_ORDERS if exists;
[hibernatetool]
[hibernatetool]     create table PURCHASE_ORDERS (
[hibernatetool]         PO_ID bigint generated by default as identity (start
with 1),
[hibernatetool]         SHIPPING_ADDRESS_STREET varchar(255),
[hibernatetool]         SHIPPING_ADDRESS_CITY varchar(255),
[hibernatetool]         BILLING_ADDRESS_STREET varchar(255),
[hibernatetool]         BILLING_ADDRESS_CITY varchar(255),
[hibernatetool]         COMMENT varchar(255),
[hibernatetool]         COMMENT_ENTERED boolean default false not null,
[hibernatetool]         primary key (PO_ID)
[hibernatetool]     );

BUILD SUCCESSFUL
Total time: 8 seconds
```

Notice in Listing 10, the SQL statement `create table PURCHASE_ORDERS`. At this point, you now have a resident database into which you can insert data with the Ant run target. So, in the third DOS console, execute the `ant run` command, as in Listing 11.

Listing 11. Running the program

```
C:\article_code\dbcode>ant run
Buildfile: build.xml

compile:

copymetafiles:

run:
     [java] Value of CommentEntered false
     [java] 1 purchase order(s) found:
     [java] Broad Street
     [java] Boston
     [java] Comment entered: true

BUILD SUCCESSFUL
Total time: 3 seconds
```

Listing 11 indicates that the database has been populated successfully. To see the contents of the database, run the `ant dbmanager` command (from yet another DOS console). You should see a window open like the one in Figure 1.

Figure 1. Database manager application

© Copyright IBM Corporation 2009. All rights reserved.

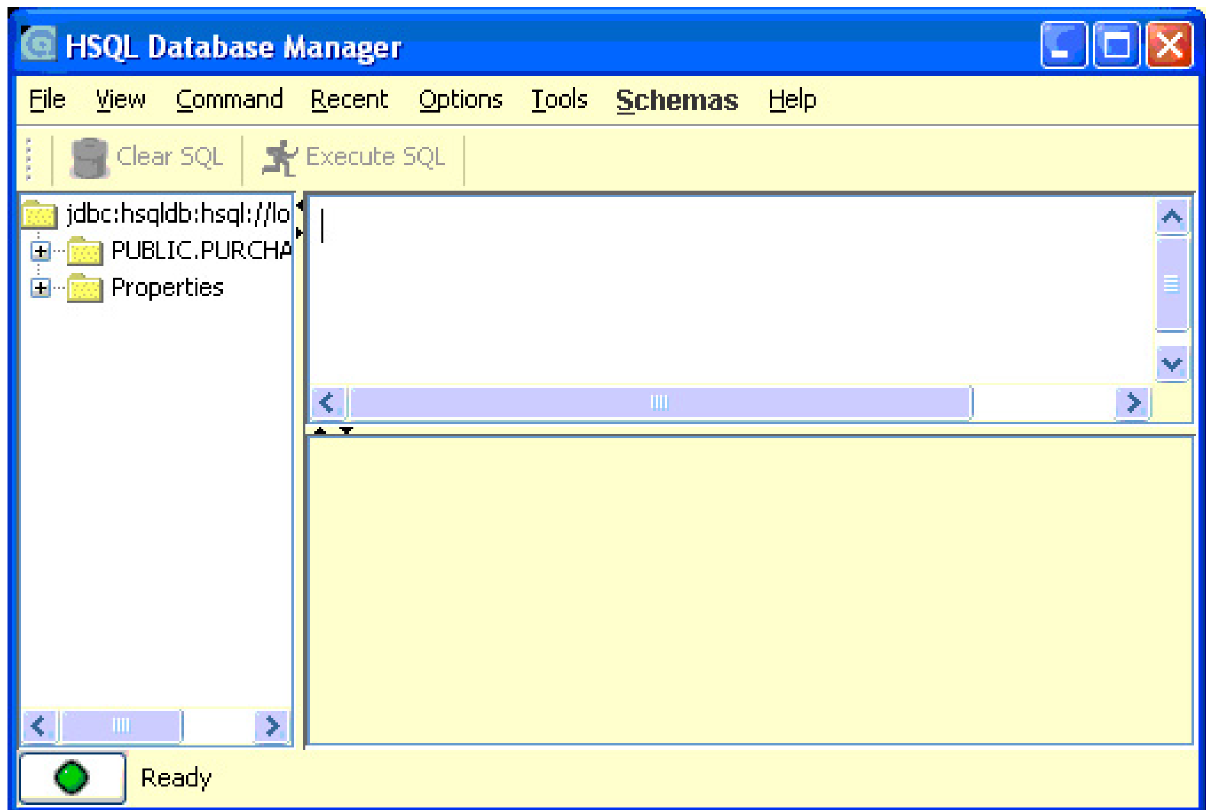


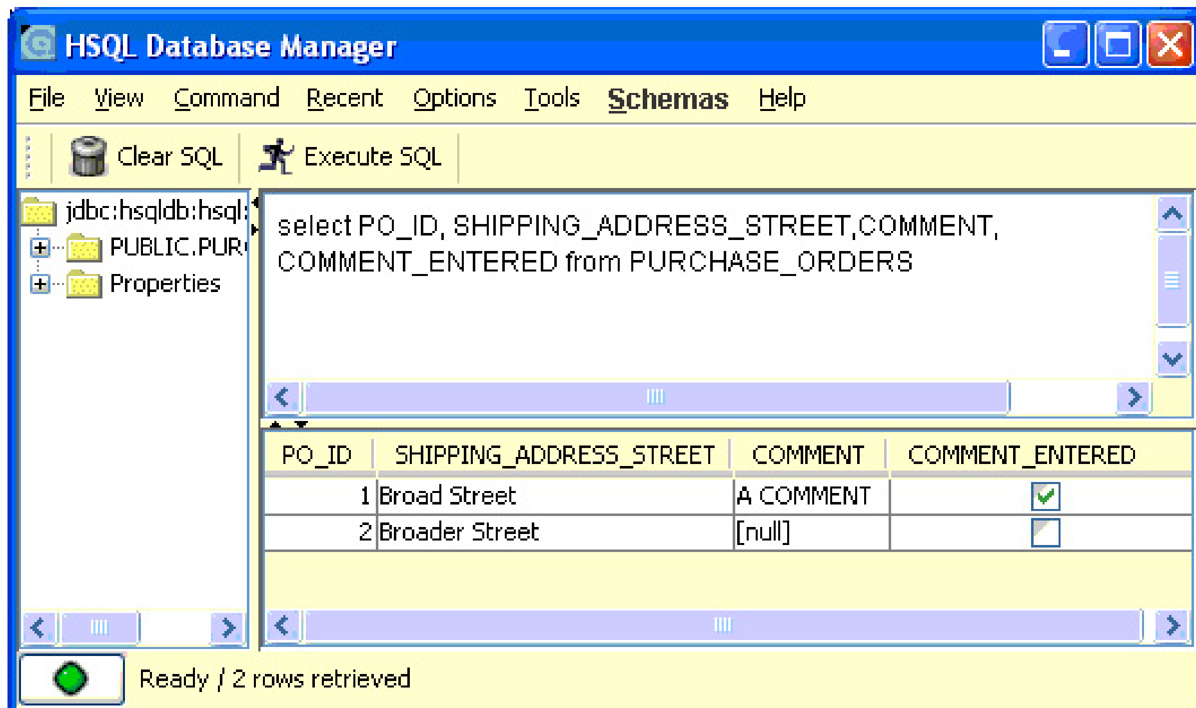
Figure 1 is a simple graphical database manager application that connects to the database you created earlier. To run a query against the database, click **Command > SELECT**, then complete the SQL statement:

```
SELECT * FROM PURCHASE_ORDERS
```

Click **Execute SQL**. You should see something like the excerpt in Figure 2.

Figure 2. The populated database

© Copyright IBM Corporation 2009. All rights reserved.



If you compare the database columns in Figure 1 with Listing 6 and Listing 7, you'll see that the Java code has in effect been subsumed into the database. This is, of course, thanks to ORM technology. The database in Figure 1 now contains two rows; Listing 12 shows the Java code that creates these two rows.

Listing 12. Persisting two objects

```
PurchaseOrder purchaseOrder = new PurchaseOrder();
purchaseOrder.setShippingAddress(new ShippingAddress("Broad Street",
"Boston"));
purchaseOrder.setBillingAddress(new BillingAddress("Broad Street", "Boston"));
purchaseOrder.setComment("A COMMENT");
purchaseOrder.setCommentEntered(true);

PurchaseOrder purchaseOrder1 = new PurchaseOrder();
purchaseOrder1.setShippingAddress(new ShippingAddress("Broader Street", "New
York"));
purchaseOrder1.setBillingAddress(new BillingAddress("Broader Street", "New
York"));

em.persist(purchaseOrder);
em.persist(purchaseOrder1);
```

The code in Listing 12 instantiates two objects of the `PurchaseOrder` class. These objects are populated, then you persist them to the database with calls to `em.persist()`. Notice that for the `purchaseOrder` object, you explicitly insert a comment; but for `purchaseOrder1`, there is no comment. With this in mind, in Figure 2, note that one row has a null entry for the **COMMENT** column. However, the corresponding **COMMENT_ENTERED** column is never null; it is either `True` or `False`.

© Copyright IBM Corporation 2009. All rights reserved.

When you want to determine whether a given row includes a comment, you no longer need to check for a null value in the **COMMENT** column. Instead, you can just read the value of the **COMMENT_ENTERED** column; if that is True, you know a value is in the **COMMENT** column. In other words, your optional XML data is correctly modeled and implemented.

Conclusion

Moving from XML data into relational data might seem a bit convoluted. You start with XSD and XML files and transform them with JAXB into corresponding Java classes. Then, you use ORM technology to populate the database. Isn't this a lot of work?

Clearly, in this article, the problem domain is very small. But if you scale things up to an average-sized enterprise application, you'll typically have a great many XML objects. These are the business domain objects, and defining them in XML makes a lot of sense. For one thing, XML definition allows for object definition by non-programmers. The other big benefit is that you can then use JAXB to flawlessly transform the XML into Java entities.

Once the XML objects are in the Java domain, the power of annotations comes to the fore, allowing you to minimally modify the Java classes to make them into persistent entities. From here, it's only a short trip to getting the data into a relational database.

Skillful use of Java object relational annotations then helps in generating an elegant schema definition. By *skillful,* I mean the judicious use of annotation elements such as `nullable=false`, which allows for only selected items to be "nullable." In conjunction with the `COLUMN_ENTERED` value, you never need to worry about null values being inserted. This is good design in action!

© Copyright IBM Corporation 2009. All rights reserved.

Downloads

Description	Name	Size	Download method
Source code for this article	code.zip	5700KB	HTTP

Information about download methods

© Copyright IBM Corporation 2009. All rights reserved.

Resources

Learn

- Optional XML in relational databases, Part 1: Are null values needed? (Stephen B. Morris, developerWorks, June 2009): Plan for optional XML elements and skip null values in your database as you explore alternative, less-invasive approaches to handle those optional XML elements that might or might not appear in XML files.
- JPA: Visit Sun's JPA wiki to learn more about the Java Persistence API for persistence and object/relational mapping for the Java EE platform.
- IBM XML certification: Find out how you can become an IBM-Certified Developer in XML and related technologies.
- XML technical library: See the developerWorks XML Zone for a wide range of technical articles and tips, tutorials, standards, and IBM Redbooks.
- Technology bookstore: Browse for books on these and other technical topics.
- developerWorks technical events and webcasts: Stay current with technology in these sessions.
- developerWorks podcasts: Listen to interesting interviews and discussions for software developers.

Get products and technologies

- Java Web Services Developer Pack 2.0: Download the developer pack to better understand the JAXB translation process shown in this article.
- Apache Ant: Download Apache Ant, a build tool using Java classes and XML.
- IBM product evaluation versions: Download or explore the online trials in the IBM SOA Sandbox and get your hands on application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.

Discuss

- XML zone discussion forums: Participate in any of several XML-related discussions.
- developerWorks blogs: Check out these blogs and get involved in the developerWorks community.

About the author

© Copyright IBM Corporation 2009. All rights reserved.

Stephen B Morris
Stephen Morris is an independent writer/consultant based in Ireland. Widely experienced in enterprise development and networking applications, Stephen has worked for some of the world's biggest networking companies on projects such as Java EE and J2SE-based network management systems, billing applications, financial systems, porting/developing SNMP entities, network device technologies, and several mobile computing applications. He holds a master's degree in computer science and holds three patents in the area of network management. He is the author of *Moving Your Career Up the Value Chain: Building Specialized Development Skills in a Global Economy* as well as numerous articles on network management and other topics.

Trademarks

IBM, the IBM logo, ibm.com, DB2, developerWorks, Lotus, Rational, Tivoli, 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.
Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.
Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.
Other company, product, or service names may be trademarks or service marks of others.

© Copyright IBM Corporation 2009. All rights reserved.