

Unmarshaling and Marshaling Data: JAXB Insurance Profile System

The Java Architecture for XML Binding (JAXB) is an implementation, geared towards making it easier to use Java applications that read, process, and output XML data. Oracle provides comprehensive support for the latest XML standards including JAXB through its Oracle XDK 10g.

The JAXB class generator that is a part of the Oracle XDK 10g allows creation of Java classes based on the XML schema. The JAXB Insurance Profile System (JAXBIPS) sample application illustrates the features of Oracle JAXB implementation. It allows the storage and retrieval of insurance profile information for different customers. This tutorial focuses on the Unmarshaling and Marshaling features of JAXB, and how it fits together in the overall scheme of the JAXBIPS.

This tutorial assumes that you are familiar with the JAXBIPS and have installed and configured the required software as described in the **Readme** and **Install** files that come with the sample download. OTN members can [download](#) complete source code and installation instructions that show how to use the JAXBIPS.

Contents

1. [Concepts](#)
2. [Design](#)
3. [Required Software](#)
4. [Setup](#)
5. [Implementation](#)
6. [Resources](#)
7. [Feedback](#)

Concepts

Understanding the following concepts will help you work through the remainder of this tutorial, and better appreciate the design and implementation decisions that OTN developers applied to the JAXBIPS sample application:

- [About JAXB](#)
- [About Unmarshaling](#)
- [About Marshaling](#)

About JAXB

Java Architecture for XML Binding (JAXB) provides API and tools that make it easier to use Java applications that read, process and output XML data. JAXB features a newer approach, and an easier way of authoring and processing XML compared to SAX or DOM, by automating the mapping between XML documents and Java objects. JAXB makes XML easy to use by compiling an XML schema into one or more Java technology classes. The combination of the schema derived classes and the binding framework enable one to perform the following operations on an XML document:

- Unmarshal XML content into a Java object representation.
- Access, update and validate the Java representation against schema constraints
- Marshal the Java representation of the XML content into XML content

Some of the advantages associated with using the JAXB framework include:

Efficient Mechanism: JAXB provides an efficient mechanism for Java developers to simplify the creation and maintenance of XML-enabled Java applications.

Increased Productivity: Java developers using JAXB are more productive because they are free from the burden of writing complex parsing code.

About Unmarshaling

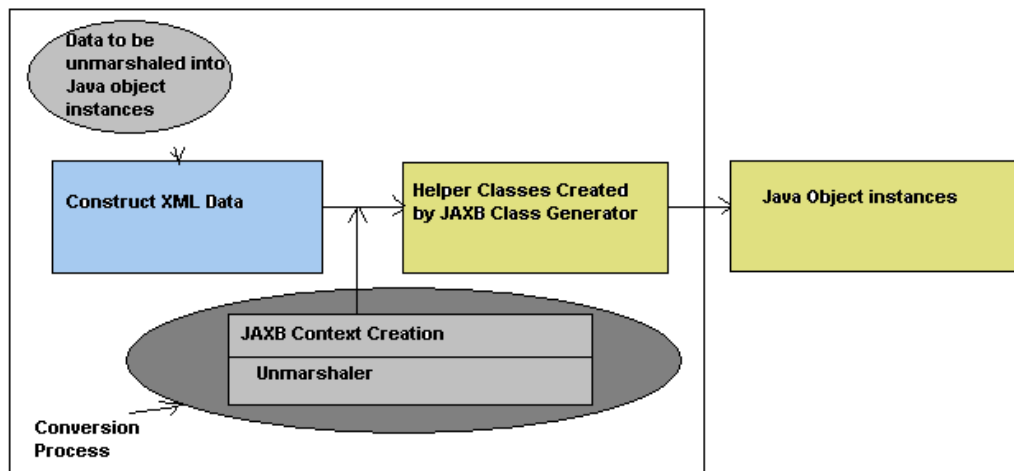
Unmarshaling, in data binding terms, refers to the conversion of XML documents to Java object instances. To better understand the concepts of Unmarshaling and Marshaling, one has to understand the key concepts behind Data binding and Class generation.

Data binding refers to the mapping of XML documents to objects and vice-versa. Other Java and XML API's (such as SAX, DOM, dom4J, JAXP, etc) achieve the same thing, however they are document-centric, in that they are driven by data in some XML representation. Data binding, on the other hand is data-centric, and allows applications to manipulate data that has been serialized as XML, thereby driving the business priority of the application.

Class generation refers to the generation of Java source files - it involves the mapping between a structure that is laid out by the constraints in an XML document and a set of Java classes. The Oracle JAXB implementation provides a JAXB class generator that takes a set of XML constraints as input and creates a set of Java source files. When these source files are compiled, you are ready to start the process of Unmarshaling.

The Marshaling and Unmarshaling API's in the java.xml.bind package within the JAXB API affects the Unmarshaling process in the Insurance Profile System application. The Streamsource API's are used for unmarshaling data from the XMLType tables in the database. The XML document to be converted should be conformant with the structure of the generated Java classes.

The block diagram below summarizes the flow of the unmarshaling process:



Unmarshaling Process Block

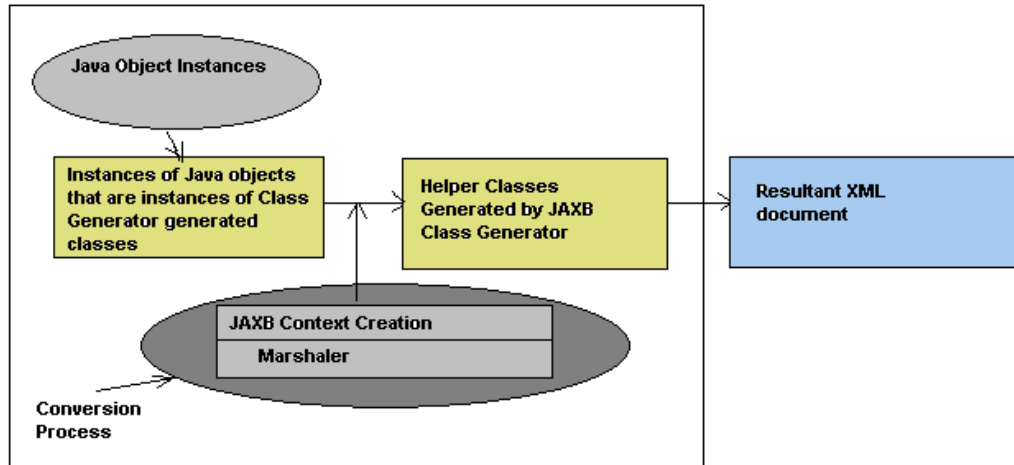
About Marshaling

Marshaling works in an opposite fashion to the Unmarshaling process - it converts a Java object into an XML document representation. Similar to the unmarshaling process, the generated Java classes must be validated, and checked for conformance with the existing Java classes before converting them into XML. As with the Unmarshaling process, the Marshaling and Unmarshaling API's in the java.xml.bind package within the JAXB API affects the marshaling process in the Insurance Profile System application.

The block diagram below summarizes the flow of the Marshaling process:

Design

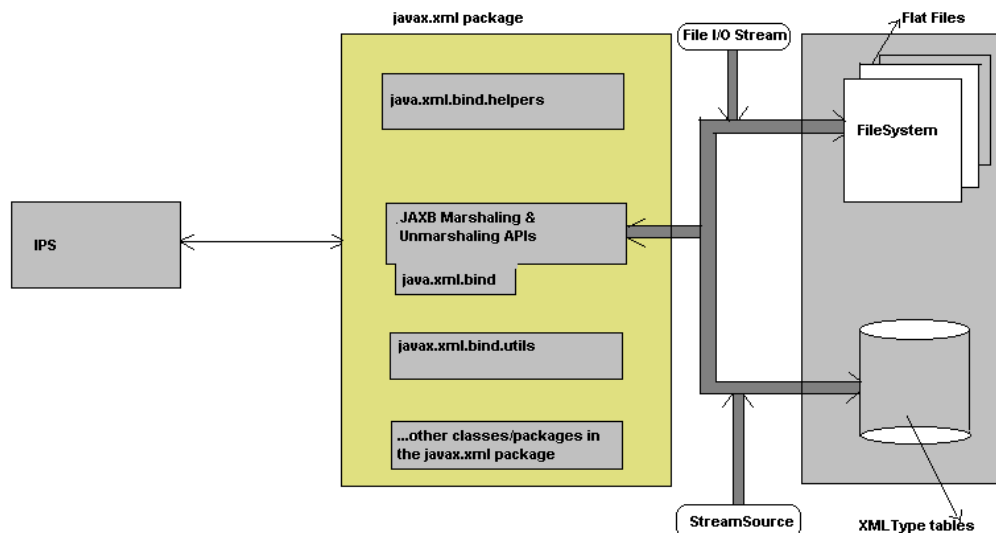
OTN developers used the Oracle JAXB implementation to effectively leverage an efficient and standard way to map between XML and Java code. Using JAXB is more productive for Java developers, in that, they can afford to write less code themselves and do not especially require XML expertise. Oracle provides comprehensive support for the latest XML standards including JAXB through its Oracle XDK 10g. The JAXB class generator part of the Oracle XDK 10g allows creation of Java classes based on the XML Schema.



Marshaling Process Block

OTN developers designed the user interface for the JAXBIPS using the Java Swing API. The database information for all the customers' insurance profile is stored in the JAXBIPS, which provides an option, of two datasources - one being the filesystem, and the other being the database. It must be noted that only one datasource can be active at any point in time. If the profile information is stored in the filesystem, then the data is organized as flat files, else, if it is stored in the XML DB, it is organized as XMLType tables.

The diagram below provides a high-level perspective of how the Marshaling and Unmarshaling elements fit together in the design of the JAXBIPS application:



When data is retrieved from the FileSystem, it is validated at the time it is loaded. In the case of data that is retrieved from the XML DB, the database inherently associates the XMLType tables to a derived schema that performs the validation. For data that is retrieved from the filesystem, its conversion into Java objects are affected using the JAXB and the File Input/Output stream API's.

In the case of data that is retrieved from the XML DB, the StreamSource API's are used for Marshaling and Unmarshaling data. JAXB provides a class generator that creates Java classes that match with the element and attribute naming in the respective XML documents.

Required Software

This tutorial presents several code examples. If you want to study them in context, download and install the [JAXBIPS source code](#). If you also want to build and run the IPS application, you will also need a Database, an IDE, a J2EE Container or an Application Server, Oracle XDK 10g and the Java Developers Kit (JDK 1.4 or later). OTN members can download developer-license versions of these Oracle products for free from <http://otn.oracle.com/software/>.

You can download complete source code and installation instructions from here:

- http://otn.oracle.com/sample_code/tech/xml/jaxb/JAXBApp.jar

To access the JAXBIPS, you will need one or more of the [Supported Clients](#).

Product	Description
Database	The JAXBIPS sample application was designed and built to run with an Oracle9i database or higher. Other databases may work, too, but have not been tested.
IDE	The JAXBIPS was built using Oracle9i JDeveloper version 9.0.3. JDeveloper provides a full-feature integrated development environment (IDE) that you can use to view and edit the source code, and to compile and deploy the application.
Oracle 10g XDK	The XML Developer's Kit for Java. You can download it from here .
Java Developer Kit	JDK 1.4 or later downloadable from here .

After installing the required software, see [Setup](#) for instructions on setting up your system and working through the tutorial.

Supported Clients

The JAXBIPS is a standalone application that uses a Java Swing based client.

Setup

Once you install and configure the *IPS sample application* as described in the steps listed below,

you can study the components described in this tutorial without any further setup. No special setup instructions are required for Unmarshaling and Marshaling Data. Listed below are the steps required to install and configure the *IPS sample application*. It contains the following sections:

- [Terminology](#)
- [Extracting the Source Code](#)
- [Configuring the Application](#)
- [Building and Running the application using Oracle 9i JDeveloper](#)

Terminology

Term	Definition
<JDEV_HOME>	The directory where Oracle9i JDeveloper is installed.
<SAMPLE_HOME>	The directory where the source files of the sample have been extracted to. For e.g., /home/.../JAXBApp, if the sample was unzipped to /home
<XDK_HOME>	The directory where Oracle Oracle XDK 10g utility is installed. For e.g., /home/.../xdk10g
<JDK_HOME>	The directory where JDK 1.4 has been installed. For e.g., /home/.../jdk1.4

Extracting the Source Code

Execute the following command to extract the files:

```
>jar xvf JAXBApp.jar
```

It is important to note that you will find the jar executable in JDK_HOME\bin. Ensure JDK_HOME\bin is present in your system path. (JDK_HOME is the root directory of the JDKx.x installation).

All the files are extracted into the JAXBApp directory.

Configuring the Application

1. Make sure that your JDK and XDK installation is configured correctly. Follow the install guides for the JDK and XDK for this purpose.

2. Open a command window if you are running on a Windows platform or a shell window if you are on Solaris or Linux system.
3. Edit your CLASSPATH and add an entry that points to <XDK_HOME>/lib/xmlparserv2.jar file. To do this, use the following commands:

On Windows:

```
d:\...\jdkjxb>set CLASSPATH=%CLASSPATH%;<XDK_HOME>\lib\xmlparserv2.jar
```

On Linux / Solaris (Note that the syntax provided here is for the bash shell)

```
$CLASSPATH=$CLASSPATH:<XDK_HOME>/lib/xmlparserv2.jar  
$export CLASSPATH
```

4. Change your current working directory to <SAMPLE_HOME>/xml.
5. On the command / shell prompt execute following command:


```
$java oracle.xml.jaxb.orajaxb -schema profile.xsd -targetPkg  
oracle.otnsamples.orajaxb -outputDir ../src
```
6. This will generate Java support classes that are used later while compiling this application.

Building and Running the Application using Oracle 9i JDeveloper

This section will describe the steps that are required to run the sample using standalone OC4J:

1. In Oracle9i JDeveloper, open the orajaxb.jws workspace that is extracted into the SAMPLE_HOME directory.
2. Expand the orajaxb.jws node by clicking it. On clicking the orajaxb.jpr will be visible. Expand orajaxb.jpr node by clicking it. Now all the java sources should be visible.
3. Right click orajaxb.jpr and select “**Project Settings...**” from pop up menu.
4. Expand "Common" node. Click on **Input Paths**. Edit java source path and type full path of “src” directory under SAMPLE_HOME.

5. Expand Configurations => Development node from left hand panel. Click on Paths node. Edit Output Directory and type full path of <SAMPLE_HOME>/classes directory under SAMPLE_HOME. This directory does not exist in the shipped jar and will be created at compile time.

6. Then Click on **Libraries** node.

7. You can view the J2SE Version that is supported with your JDeveloper in the right hand panel. Note: For this application J2SE version 1.4 or later is required. Hence you need to define a J2SE 1.4 for this project. For doing so, please use the following steps:
 - i. Click on the Define button. The New J2SE dialog window pops up.
 - ii. Enter "J2SE Name " as JDK1.4 and then click on the Browse button for J2SE Executable entry.
 - iii. Select java executable from the JDK_HOME\bin
 - iv. Click OK on the New J2SE Window and return to the Project settings main window.
 - v. From the "Available Libraries" box in the right hand panel select the JDeveloper Runtime and click on single right-handed arrow in the middle. This will add the library in the current project. Similarly, select Oracle JDBC library and add it to current project.

8. From selected libraries pane select xmlparserv2 library. Click on Edit button.
 - i. Edit library dialog pops up.
 - ii. Click on Edit button for CLASSPATH entry.
 - iii. Edit Class Path dialog pops up. Remove existing entry and add new entry that points to xmlparserv2.jar from XDK_HOME\lib. Press OK button to dismiss the box.
 - iv. Click OK button to dismiss all open windows.

9. Return to the System Navigator pane on the left hand side.

10. Click on the Green + sign in the left hand corner. This will pop up Add Files or Directories dialog box.

11. Go to <SAMPLE_HOME>/src/oracle/otnsamples/orajaxb/profile directory.

12. Select all java files and jaxb.properties file and add these files to orajaxb.jpr project.

13. To compile the java sources, right click on orajaxb.jpr and select "Rebuild orajaxb.jpr" option. This should successfully compile all the java sources. Oracle9i JDeveloper creates the classes inside <SAMPLE_HOME>/classes directory with the relevant package structure.

14. Select MainFrame.java by clicking on the filename in System Navigator panel. For running the application, right click MainFrame.java and select "**Run MainFrame.java**" option.

For more details on how to use the application, please refer to the Readme file that comes with the JAXBApp.jar.

Implementation

The JAXB Insurance Profile System (JAXBIPS) sample application implements several features defined in the Oracle JAXB specifications, including:

- Usage of the JAXB class generator

- Usage of the Marshaler and Unmarshaler classes

- Marshaling and Unmarshaling from File and StreamSource

- Using JAXB in conjunction with XMLType

- Binding of XML Schema and Complex types

- Using collection types in schema definition (Multiple claim information)

- Unmarshal-time validation

This tutorial highlights one of the most important features of JAXB - Unmarshaling and Marshaling Data. In this section we will look at how the design decisions for the JAXBIPS are implemented in the application, by looking at related code snippets. Before writing code, we need to generate support classes with the help of the Oracle JAXB compiler that is part of the oracle.xml.jaxb package. In order for us to use this, we need to have the xmlparserv2.jar in the CLASSPATH:

On the command / shell prompt execute following command:

```
$java oracle.xml.jaxb.orajaxb -schema profile.xsd -targetPkg oracle.otnsamples.orajaxb  
-outputDir ../src
```

To describe the XML document structure, we need to have a schema definition file. Using profile.xsd we build the classes that allow us to marshal, unmarshal and validate XML document instances against the schema definition file. The Schema definition file in our case is the [profile.xsd](#).

Marshaling and Unmarshaling - FileSystem

We will now look at the code that marshals the Java instances into the File System:

```

public void saveRecord(Record rec) {
    try {
        //Create a JAXBContext instance
        JAXBContext jc = JAXBContext.newInstance(
            "jaxbderived.profile"
        );
        String filename = null;
        if ( !dirPath.endsWith(".xml"))
            filename = dirPath + "/" + currentRecord.getCustomerID() + ".xml";
        else filename = dirPath;

        textArea.append(" Saving file " + filename + "\n");

        File f = new File(filename);

        f.delete();

        //Create a marshaller object
        Marshaller m = jc.createMarshaller();
        StringWriter swriter = new StringWriter();

        //Marshal the record object
        m.marshal(rec, swriter);

        //Store the marshaled object in the String
        String s = swriter.toString();

        //Output this String to a file
        FileOutputStream fout = new FileOutputStream(filename);
        PrintWriter pw = new PrintWriter(fout);

        pw.write(s);

        //Flush and close file and printWriter
        pw.flush();

        if (pw != null) {
            pw.close();
        }

        if (fout != null) {
            fout.close();
        }

        textArea.append("Record saved... \n");
    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

Here's the code that unmarshals the document instances from the File System:

```
public void loadSingleRecord() {
    textArea.append("This may take some time. Please wait... \n");

    try {
        //Create a JAXBContext instance
        JAXBContext jc = JAXBContext.newInstance(
            "jaxbderived.profile"
        );

        //Create an unmarshaller instance
        Unmarshaller u = jc.createUnmarshaller();
        u.setValidating(true);
        textArea.append("\nReading file " + dirPath + "\n");

        //Unmarshal the Record object from FileInputStream
        Record rec = (Record) u.unmarshal(fileToURL(dirPath));

        //Store the object in the list
        list.add(rec);
        textArea.append("One record loaded.\n Size of the list is " +
            list.size() + " \n"
        );
    } catch (Exception ex) {
        textArea.append("There was an error while parsing \n" + dirPath +
            "\nThis file will be ignored. \n"
        );
        textArea.append(
            "Please note that this error is thrown when the source XML does not "
        );
        textArea.append(
            "confirm to the Profile Record \nSchema definition. The error is given below \n"
        );
        textArea.append(ex.toString() + "\n\n");
    } finally {
    }
}
```

Marshaling and Unmarshaling - Database

Here's the code to marshal and save the records into the database:

```

public void saveRecord(Record rec) {
    Statement st = null;
    JAXBContext jc = null;
    Marshaller m = null;
    OraclePreparedStatement stmt = null;

    try {
        st = con.createStatement();
        textArea.append("Saving record with customer id " +
            rec.getCustomerID() + " \n"
        );

        //Delete previous record if any
        st.execute(
            "DELETE profile_tab pt WHERE pt.c1.extract('/Record/@CustomerID').getStringVal()
=" +
            rec.getCustomerID() + ""
        );

        if (st != null) {
            st.close();
        }

        //Insert updated record
        stmt = (OraclePreparedStatement) con.prepareStatement(
            "INSERT INTO profile_tab VALUES
            (XMLType.CreateXML(?).CreateSchemaBasedXML
('http://jaxbderived.profile/profile.xsd'))"
        );

        //Create JAXBContext instance
        jc = JAXBContext.newInstance("jaxbderived.profile");

        //Create Marshaller instance
        m = jc.createMarshaller();

        StringWriter swriter = new StringWriter();

        //Marshal record object to a StringWriter
        m.marshal(rec, swriter);

        //Set the parameter
        stmt.setString(1, swriter.toString());

        //Execute the insert statement
        stmt.execute();
        con.commit();
        textArea.append("One record has been added / updated \n");
    }
}

```

Here is the code that unmarshals the document instances from the database:

```

private void createListFromDatabase() {
    // Create a list instance
    list = Collections.synchronizedList(new ArrayList());

    Statement st = null;
    ResultSet rs = null;
    Unmarshaller u = null;
    JAXBContext jc = null;

    try {
        //Create a JAXBContext instance
        jc = JAXBContext.newInstance("jaxbderived.profile");

        //Create unmarshaller object
        u = jc.createUnmarshaller();
        u.setValidating(false);
        textArea.append("Reading records from the database \n");
        st = con.createStatement();

        //Select all records from profile_tab
        rs = st.executeQuery(
            "SELECT pt.c1.getStringVal() FROM profile_tab pt"
        );

        int i = 0;

        while (rs.next()) {
            //Retrieve the XMLType object as a String
            String s = rs.getString(1);

            //Get byte array from the String object
            byte[] bytes = s.getBytes();

            //Create ByteArrayInputStream object
            ByteArrayInputStream bais = new ByteArrayInputStream(bytes);

            try {
                //Unmarshal the record
                Record rec = (Record) u.unmarshal(new StreamSource(bais));
                //Add the record to the list
                list.add(rec);
            } catch (Exception e1) {
                textArea.append("There was an error while parsing " + i +
                    " th record from the result set \n" +
                    " This record has been skipped"
                );
            }
        }

        i++;
    }
}

```

Resources

Following are links to resources that can help you understand and apply the concepts and techniques presented in this tutorial. See the [Required Software](#) section to obtain the JAXBIPS source code and related files:

Resource	URL
Insurance Profile System using JAXB	http://otn.oracle.com/sample_code/tech/xml/jaxb/index.html
Oracle Products	http://otn.oracle.com/products/
OTN Sample Code	http://otn.oracle.com/sample_code/content.html
XML Tech Center	http://www.otn.oracle.com/tech/xml/content.html
JAXB Specification	http://java.sun.com/xml/downloads/jaxb.html

Feedback

If you have questions or comments about this tutorial, you can:

- Post a message in the [OTN Sample Code discussion forum](#). OTN developers and other experts monitor the forum.
- Send email to the author. <mailto:Dilip.Thomas@oracle.com>

If you have suggestions or ideas for future tutorials, you can:

- Post a message in the [OTN Member Feedback forum](#).
 - Send email to <mailto:Tom.Hunert@oracle.com>
-