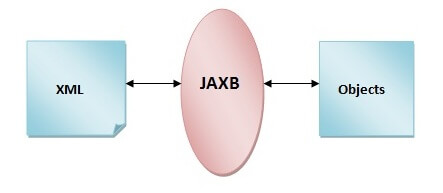
**JAXB** provides concepts and API to convert object into XML and XML into object.

**JAXB** stands for Java Architecture for XML Binding.

It provides mechanism to marshal (write) java objects into XML and unmarshal (read) XML into object. Simply, you can say it is used to convert java object into xml and vice-versa.



## **Features of JAXB 2.0 :**

JAXB 2.0 includes several features that were not present in JAXB 1.x. They are as follows:

**1) Annotation support**: JAXB 2.0 provides support to annotation so less coding is required to develop JAXB application. The javax.xml.bind.annotation package provides classes and interfaces for JAXB 2.0.

**2) Support for all W3C XML Schema features**: it supports all the W3C schema unlike JAXB 1.0.

**3) Additional Validation Capabilities**: it provides additional validation support by JAXP 1.3 validation API.

**4) Small Runtime Library**: it required small runtime library that JAXB 1.0.

**5) Reduction of generated schema-derived classes**: it reduces a lot of generated schema-derived classes.

## **Simple JAXB Marshalling Example: Converting Object into XML :**

Let's see the steps to convert java object into XML document.

* Create POJO or bind the schema and generate the classes
* Create the JAXBContext object
* Create the Marshaller objects
* Create the content tree by using set methods
* Call the marshal method

**@XmlRootElement** specifies the root element for the xml document.

**@XmlAttribute** specifies the attribute for the root element.

**@XmlElement** specifies the sub element for the root element.

import javax.xml.bind.annotation.XmlAttribute;

import javax.xml.bind.annotation.XmlElement;

import javax.xml.bind.annotation.XmlRootElement;

**@XmlRootElement**

public class Employee {

private int id; private String name; private float salary;

public Employee() {}

public Employee(int id, String name, float salary) {

this.id = id;

this.name = name;

this.salary = salary;

}

**@XmlAttribute**

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

**@XmlElement**

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

**@XmlElement**

public float getSalary() {

return salary;

}

public void setSalary(float salary) {

this.salary = salary;

}

}

import java.io.FileOutputStream;

import javax.xml.bind.JAXBContext;

import javax.xml.bind.Marshaller;

public class ObjectToXml {

public static void main(String[] args) throws Exception{

JAXBContext contextObj = JAXBContext.newInstance(Employee.class);

Marshaller marshallerObj = contextObj.createMarshaller();

marshallerObj.setProperty(Marshaller.JAXB\_FORMATTED\_OUTPUT, true);

Employee emp1=new Employee(1,"Vimal Jaiswal",50000);

marshallerObj.marshal(emp1, new FileOutputStream("employee.xml"));

}

}

Output:

The generated xml file will look like this:

File: employee.xml

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<employee id="1">

<name>Vimal Jaiswal</name>

<salary>50000.0</salary>

</employee>

**Simple JAXB UnMarshalling Example: Converting XML into Object**

File: XMLToObject.java

import java.io.File;

import javax.xml.bind.JAXBContext;

import javax.xml.bind.JAXBException;

import javax.xml.bind.Unmarshaller;

public class XMLToObject {

public static void main(String[] args) {

try {

File file = new File("employee.xml");

JAXBContext jaxbContext = JAXBContext.newInstance(Employee.class);

Unmarshaller jaxbUnmarshaller = jaxbContext.createUnmarshaller();

Employee e=(Employee) jaxbUnmarshaller.unmarshal(file);

System.out.println(e.getId()+" "+e.getName()+" "+e.getSalary());

} catch (JAXBException e) {

e.printStackTrace();

}

}

}

**What binding stands for JAXB?**

Jakarta XML **Binding** (**JAXB**; formerly **Java** Architecture for XML **Binding**) is a software framework that allows Jakarta EE developers to map **Java** classes to XML representations. **JAXB** provides two main features: the ability to marshal **Java** objects into XML and the inverse, i.e. to unmarshal XML back into **Java** objects.

**Can Jaxb used for JSON?**

EclipseLink **JAXB** (MOXy) is one of **JAXB** implementation which is mostly **used** to create java classes from XML or **JSON**. In Java **JAXB** provides two general purpose implementation. Marshalling – It Converts a Java object into XML or **JSON**. Unmarshalling – It Converts XML or **JSON** into a Java Object.

**JAXB JSON Example :**  [JAXB](https://jaxb.java.net/" \o "JAXB" \t "https://examples.javacodegeeks.com/core-java/xml/bind/jaxb-json-example/_blank) is a java architecture for XML binding is an efficient technology to convert XML to and from Java Object. [EclipseLink JAXB (MOXy)](https://www.eclipse.org/eclipselink/moxy.php" \o "MOXy" \t "https://examples.javacodegeeks.com/core-java/xml/bind/jaxb-json-example/_blank) is one of [JAXB](https://jaxb.java.net/" \o "JAXB" \t "https://examples.javacodegeeks.com/core-java/xml/bind/jaxb-json-example/_blank) implementation which is mostly used to create java classes from XML or JSON. In Java [JAXB](https://jaxb.java.net/" \o "JAXB" \t "https://examples.javacodegeeks.com/core-java/xml/bind/jaxb-json-example/_blank) provides two general purpose implementation.

**Marshalling –** It Converts a Java object into XML or JSON.

**Unmarshalling –** It Converts XML or JSON into a Java Object.

**Example:**

**Step 1 : Add dependency in pom.xml**

<dependencies>

<dependency>

<groupId>org.eclipse.persistence</groupId>

<artifactId>org.eclipse.persistence.moxy</artifactId>

<version>2.5.2</version>

</dependency>

<dependency>

<groupId>javax.xml.bind</groupId>

<artifactId>jaxb-api</artifactId>

<version>2.2.11</version>

</dependency>

</dependencies>

**Step 2 : Simple Pojo: Create an employee object, initialized with some values, it will be converted to / from JSON.**

package com.jcg.jaxb.json;

import java.util.List;

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement

public class Employee {

private int id;

private String name;

private List skills;

// setter & getter

}

**Step : 3 : Marshal Java Object to JSON:**

Create a JaxBContext using the Employee class then convert the “employee” Java object into JSON formatted string using Marshaller object with following three properties:

* **MEDIA\_TYPE –** Determine the produced output media type (JSON, XML).
* **JSON\_INCLUDE\_ROOT –** Flag to determine whether you want to include the JSON root element in the produced output or not.
* **JAXB\_FORMATTED\_OUTPUT –** Flag to determine whether you want to format the produced output or not.

public class MarshallerDemo {

/\*\*

\* @param args

\* @throws JAXBException

\* Marshaller POJO to JSON using EclipseLink MOXy

\*/

public static void main(String[] args) throws JAXBException {

// Creating a new employee pojo object with data

Employee employee = new Employee();

employee.setId(1);

employee.setName("Ashraf");

List skills = new ArrayList();

skills.add("java");

skills.add("sql");

employee.setSkills(skills);

// Create a JaxBContext

JAXBContext jc = JAXBContext.newInstance(Employee.class);

// Create the Marshaller Object using the JaxB Context

Marshaller marshaller = jc.createMarshaller();

// Set the Marshaller media type to JSON or XML

marshaller.setProperty(MarshallerProperties.MEDIA\_TYPE,

"application/json");

// Set it to true if you need to include the JSON root element in the JSON output

marshaller.setProperty(MarshallerProperties.JSON\_INCLUDE\_ROOT, true);

// Set it to true if you need the JSON output to formatted

marshaller.setProperty(Marshaller.JAXB\_FORMATTED\_OUTPUT, true);

// Marshal the employee object to JSON and print the output to console

marshaller.marshal(employee, System.out);

}

}

**OUTPUT:**

{

"employee" : {

"id" : 1,

"name" : "Ashraf",

"skills" : [ "java", "sql" ]

}

}

**Step :3\_1. Unmarshal JSON to Java Object:**

Create a JaxBContext using the Employee class then read the provided JSON string and convert it back to the “employee” Java object using Unmarshaller object with following two properties:

* **MEDIA\_TYPE – D**etermine the provided input media type (JSON, XML).
* **JSON\_INCLUDE\_ROOT –** Flag to determine whether you want to include the JSON root element in the provided input or not.

public class UnmarshallerDemo {

/\*\*

\* @param args

\* @throws JAXBException

\* Unmarshaller JSON to POJO using EclipseLink MOXy

\*/

public static void main(String[] args) throws JAXBException {

// Create a JaxBContext

JAXBContext jc = JAXBContext.newInstance(Employee.class);

// Create the Unmarshaller Object using the JaxB Context

Unmarshaller unmarshaller = jc.createUnmarshaller();

// Set the Unmarshaller media type to JSON or XML

unmarshaller.setProperty(UnmarshallerProperties.MEDIA\_TYPE,

"application/json");

// Set it to true if you need to include the JSON root element in the

// JSON input

unmarshaller

.setProperty(UnmarshallerProperties.JSON\_INCLUDE\_ROOT, true);

// Create the StreamSource by creating StringReader using the JSON input

StreamSource json = new StreamSource(

new StringReader(

"{\"employee\":{\"id\":1,\"name\":\"Ashraf\",\"skills\":[\"java\",\"sql\"]}}"));

// Getting the employee pojo again from the json

Employee employee = unmarshaller.unmarshal(json, Employee.class)

.getValue();

// Print the employee data to console

System.out.println("Employee Id: " + employee.getId());

System.out.println("\nEmployee Name: " + employee.getName());

System.out.println("\nEmployee Skills: "

+ StringUtils.join(employee.getSkills(), ','));

}

}

The class contains the following annotations:

**@XmlRootElement:** the name of the root XML element is derived from the class name and we can also specify the name of the root element of the XML using its name attribute

**@XmlType:** define the order in which the fields are written in the XML file

**@XmlElement:** define the actual XML element name which will be used

**@XmlAttribute:** define the id field is mapped as an attribute instead of an element

**@XmlTransient:** annotate fields that we don't want to be included in XML

Example :

@XmlRootElement(name = "book")

@XmlType(propOrder = { "id", "name", "date" })

public class Book {

private Long id; private String name; private String author;

private Date date;

@XmlAttribute

public void setId(Long id) {

this.id = id;

}

@XmlElement(name = "title")

public void setName(String name) {

this.name = name;

}

@XmlTransient

public void setAuthor(String author) {

this.author = author;

}

// constructor, getters and setters

}