

Java Jazz up

A BETTER WAY TO LEARN PROGRAMMING

**ORM
FRAMEWORK**

Apache Axis

XML Parsers



Tips & Tricks / Advertise With Us

India's Cheapest web Service Provider

 **RoseIndia**

OUR SERVICES

- **ERP Soluiotns**
- **Software Solutions**
- **Web Development**
- **Web Designing**
- **Web Redesigning**
- **Domain Registration**
- **Web Promotion**
- **SEO**
- **Article Writing**
- **Blog Writing**
- **News Writing**
- **SEO Copywriting**
- **Technical Documentation**
- **E-Commerce Solutions**
- **CRM**
- **Outsourcing**

Extend your reach
with our solutions...

<http://www.roseindia.net/services>

"Developing plan is actually laying out a sequence of events that have to occur for you to achieve your goal."

Published by

RoseIndia

JavaJazzUp Team

Editor-in-Chief

Deepak Kumar

Editor-Technical

Ravi Kant

Sr. Graphics Designer

Suman Saurabh

Graphics Designer

Santosh Kumar
Amardeep Patel

**Register with JavaJazzUp
and grab your monthly issue**

"Free"

Editorial

Dear Readers,

We are back here with the Feb 2008 issue of Java Jazz-up. The current edition is designed in continuation with the previous issues focusing the sprouting technocrats. This issue highlights the interesting Java technologies especially for the beginners.

Though it was a hard job to simplify the complexities of the technologies like Hibernate 3.0, EJB 3.0, struts 2, JSF and Design Patterns. Still our team has done a marvelous work in making it easy and simpler for the new programmers regime. This issue reflects our consistent attempts to avail the quality technological updates that enforce the readers to appreciate it a lot and be a part of its Readers Community.

Java News and Updates section provides the latest updates of the things happening around the globe making the readers aware of the java technological advancement. In this section, you will know the new features introduced in the existing tools, utilities, application servers, IDEs, along with the Java API updates.

The set of articles conferring technologies like Design patterns, JSF, Hibernate 3.0, etc. are provided in such a manner that even a novice learns and implements the concepts in a very easy manner.

We are providing it in a PDF format so that you can view and even download it as a whole and get its hard copy.

Please send us your feedback about this issue and participate in the Reader's Forum with your problems, issues concerned with the topics you want us to include in our next issues.

Editor-in-Chief

Deepak Kumar
Java Jazz up

Content

- 05** [Java News](#) | Sun Microsystems has concurred all suggestions complying with the idea, to buy MySQL AB for \$1B, just by providing them additional influence in the open source community and providing access to MySQL to its larger corporations.

- 07** [Working with Entity bean using JPA API](#) | In the previous issue of Java Jazz Up you have learnt how a Stateless Bean is developed in EJB 3.

- 16** [XML Schema](#) | XML Schema is a W3C Standard. It is an XMLbased alternative to DTDs. It describes the structure of an XML document.

- 23** [XML- SAX Parser using JAXP API](#) | In the previous issue of Java Jazz Up you have read about XML technology. In this issue, you will learn how XML technology works with Javausing different kinds of XML parsers.

- 31** [Hibernate Query Language](#) | Hibernate Query Language or HQL for short is extremely powerful query language. HQL is much like SQL and are case-insensitive, except for the names of the Java Classes and properties.

- 36** [Web Services - Apache Axis](#) | Axis stands for Apache's EXtensible Interaction System. Axis is one of the most popular SOAP engine that is used to construct SOAP processor like gateways, clients, servers etc.

- 37** [Struts2 Tags](#) | not only let the developers use converters provided already by the implementation but also create their own converters according to the requirement of the application. This topic explains about how to create custom converter.

- 54** [JSF Application](#) | Event Handling is one of the important concepts in JSF. This section provides a simple JSF application, which explains how to implement "immediate" event handling in JSF.

- 59** [Design Pattern](#) | This design pattern defines one to many dependency between the objects so that if an object changes its state then all the dependent objects are notified and updated automatically.

- 63** [Tips & Tricks](#) | This example retrieves data from MySQL and sends response to the web browser in the form of a PDF document using Servlet. This program uses iText, which is a java library containing classes to generate documents in PDF, XML,HTML, and RTF.

- 69** [Advertise with Us](#) | We are the top most providers of technology stuffs to the java community.

- 70** [Valued JavaJazzup Readers Community](#) | We invite you to post Java-technology oriented stuff. It would be our pleasure to give space to your posts in JavaJazzup.

Java News and Releases

Sun Microsystems to buy MySQL AB for \$1B

Sun Microsystems has concurred all suggestions complying with the idea, to buy MySQL AB for \$1B, just by providing them additional influence in the open source community and providing access to MySQL to its larger corporations. They are looking forward to close the deal during the third or fourth quarter.

Hybrid Java Profiling - New Profiling Strategies

This new approach of Java profiling merges the advantages of sampling as well as event profiling just to make it easy, extensible, and more influential runtime performance just to monitor the solutions. This helps to scale from development to production. It also simplifies the way to achieve the high precision accuracy from event profiling having the low overhead of sampling.

Shades of HotJava: LoboBrowser, a web browser in Java

Lobo is being developed actively in order to fulfill the aim to provide the fully support Javascript, HTML 4 and CSS2. LoboBrowser, is not a HotJava but a full-Java web browser that runs JavaScript.

It generally used to produce fast, complete, easy to extend, featured-rich and secure browser. It also works as a interpretation engine, through which we can also watch the DOM of a page even after JavaScript has been run over it.

Desiderata Releases Jaxcent (Java AJAX) for the Internet

The recent released Jaxcent V2, Jaxcent V2, recently released, furnishes the full Java AJAX Programming capabilities from the server side. It is not mandatory to work with JavaScript, standard servlet containers, but add an include statement to HTML content. A Java class implements the JavaScript capabilities, while the client-side puts into use the server-side class just to provide the full functional implementation.

DigitalPerson offers fingerprint recognition for Java-based POS terminals

DigitalPerson offers a new fingerprint recognition software for a new generation of point-of-sale terminals, developed in java, enabling POS terminal developers to quickly and easily add accountability and security capabilities. It provides improved profitability of their customers by enabling employee time-and-attendance and anti-theft features.

Sun releases Java SE 6 Update 4

Sun has released Java SE 6 Update 4. This update fixes 370 bugs including several security issues, also fixes several flaws that can result into system crashes. It includes JAX-WS 2.1 API in the rt.jar so you no longer you need to copy JAX-WS or JAXB API jars in JAVA_HOME/jre/lib/endorsed.

India's Cheapest
web Service
Provider

Web: <http://www.roseindia.net/services/> E-mail: deepak@roseindia.net

 **RoseIndia**
Technologies Pvt. Ltd.

Extend your reach

with our solutions...

JBoss Enterprise Application Platform 4.3 released

The JBoss team at Red Hat has announced the release of JBoss Enterprise Application Platform 4.3 having features of upgraded messaging and Web services technologies.


It integrates JBoss Application Server with hibernate ORM software and Seam application framework for building Web 2.0 applications. JBoss Messaging acts as the messaging architecture and JBoss Web Services supports JAX-WS) and Web services API.

JVM Lies: The OutOfMemory Myth

The OutOfMemory Myth," discourses about what happens when a JVM throws an OutOfMemoryError – most of the programmers have noticed that, it may indicates out of memory, but it doesn't look like this always and throws more RAM at the JVM may help, but that's is not the right solution.


CodeGear(TM) JBuilder(R) 2007 Is Named Best Java IDE

InfoWorld has named CodeGear JBuilder 2007 for "Best Java Integrated Development Environment" of the year 2007. JBuilder 2007 IDE is built on the Eclipse framework that is used to develop Java and Web-based applications more fast and reliable. "I found a very smooth, very robust IDE with many innovative features. It's safe to say that CodeGear decided to throw everything it had at this release -- and succeeded brilliantly," wrote Andrew Binstock, senior contributing editor at InfoWorld.



8 × 3 = 24hours

It's always for 24h



NEWSTRACK india

Working with Entity bean using JPA

Working with Entity bean using JPA API

In the previous issue of Java Jazz Up you have learnt how a Stateless Bean is developed in EJB 3. Now you will learn how an Entity Bean is developed to interact with the Database using JPA API.

Entity:

An entity defines a table (consisting of rows and columns) in a relational database. An entity refers to a logical collection of data that can be stored or retrieved from that entity. For example, in a banking application, **Customer** and **BankAccount** can be treated as entities. Customer name, customer address etc can be logically grouped together for representing a Customer entity. Similarly account number, total balance etc may be logically grouped under BankAccount entity.

Mapping with EJB3/JPA Annotations:

The EJB 3.0 entity beans are used to model and access relational database tables. It is a completely POJO-based persistence framework with annotations that specify how the object should be stored in the database. The EJB 3.0 container does the mapping from the objects to relational database tables automatically and transparently. Their mappings are defined through JDK 5.0 annotations (an XML descriptor syntax for overriding is defined in the EJB3 specification).

Annotations can be split in two categories, the **logical mapping** annotations which allows programmer to describe the object model, the class associations, etc. and the **physical mapping** annotations which describes the physical schema, tables, columns, indexes, etc. The combination of annotations from both categories makes a **JPA**-based application. Now, the Java developer no longer needs to worry about for the implementation of **home interfaces** and the details of the database table schema, database connection management, and specific database access APIs.

Introduction to Java Persistence API (JPA)

Java Persistence API is a lightweight framework based on POJO for object-relational mapping. It is the standard API added in Java EE 5 platform and used for the management of the persistent data and object/relational mapping. Persistence that deals with storing and retrieving of application data can now be programmed with Java Persistence API starting from EJB 3.0. Every application server compatible with Java EE 5 supports the Java Persistent APIs.

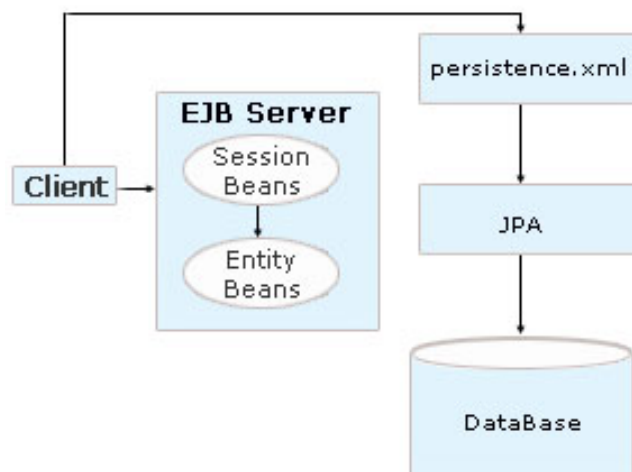
Java language metadata **annotations** and/or XML deployment descriptor is used for the mapping between Java objects and a relational database.

The Java Persistence API contains the following areas:

- Java Persistence API
- O-R mapping metadata
- The query language

Here we are describing EJB3-JPA by using the simple domain object model by an example.

Working process of an EJB application using JPA:



While developing an enterprise application, first design the domain object model required

Working with Entity bean using JPA

to persist the data in the database. **The Database Model is called the Domain Model** that represents the persistence objects or entities in the database. An entity may be a person, place or a thing about which you want to store the data in the database. A rich domain model includes the characteristics of all the object-oriented behavior like inheritance, polymorphism and many more.

Lets read about the some common annotations that are used to develop a JPA application.

Primary Key Generation:

In EJB 3.0, a primary key is used with **@Id** annotation. Depending upon the application requirement, Id annotation can be used with different primary key generation strategies defined by **GeneratorType** enum. The GeneratorTypes are TABLE, SEQUENCE, IDENTITY, AUTO, and NONE.

Declaring an entity bean:

Every bound persistent POJO class is an entity bean and is declared using the **@Entity** annotation (at the class level). **@Entity** declares the class as an entity bean (i.e. a persistent POJO class), which tells the EJB3 container that this class needs to be mapped to a relational database table.

Defining the table:

@Table is set at the class level; it allows you to define the table, catalog, and schema names for your entity bean mapping. If no **@Table** is defined the default values are used, that is the unqualified class name of the entity.

Here is the sample code using of these annotations:

```
@Entity
@Table(name="book")
@SequenceGenerator(name =
"book_sequence", sequenceName =
"book_id_seq")

public class Book implements Serializable {
Long empid;
```

```
@Id
@GeneratedValue
public Long getId()
{ return id; }
public void setId(Long id)
{ this.id = id; }
}
```

The **@Table** defines the table name. Each instance of the entity bean represents a row of data in the table. Each column in the table corresponds to a data attribute in the entity bean. The **@SequenceGenerator** defines a sequence generator. A sequence is a database feature. It returns the next Integer or Long value each time it is called.

@Id declares the identifier property of this entity bean. **@GeneratedValue** annotation indicates that the server automatically generates the primary key value.

Managing Entities:

The entity manager manages entities. The entity manager is represented by `javax.persistence.EntityManager` instances. Each `EntityManager` instance is associated with a persistence context.

A persistence context defines the scope under which particular entity instances are created, persisted, and removed. It is a set of managed entity instances that exist in a particular data store. The `EntityManager` interface defines the methods that are used to interact with the persistence context.

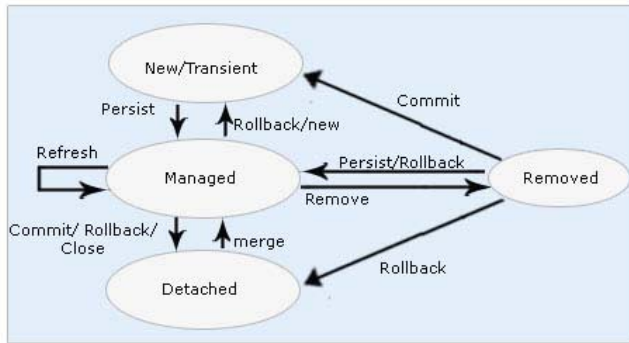
The EntityManager Interface:

The `EntityManager` API creates and removes persistent entity instances, finds entities by the entity's primary key, and allows queries to be run on entities. To obtain an `EntityManager` instance, inject the entity manager into the application component:

```
@PersistenceContext
EntityManager em;
```


Working with Entity bean using JPA

Managing an Entity Instance's Life Cycle:



Persistence Content

You manage entity instances by invoking operations on the entity by means of an EntityManager instance. Entity instances are in one of four states: new, managed, detached, or removed.

New entity instances have no persistent identity and are not yet associated with a persistence context.

Managed entity instances have a persistent identity and are associated with a persistence context.

Detached entity instances have a persistent identity and are not currently associated with a persistence context.

Removed entity instances have a persistent identity, are associated with a persistent context, and are scheduled for removal from the data store.

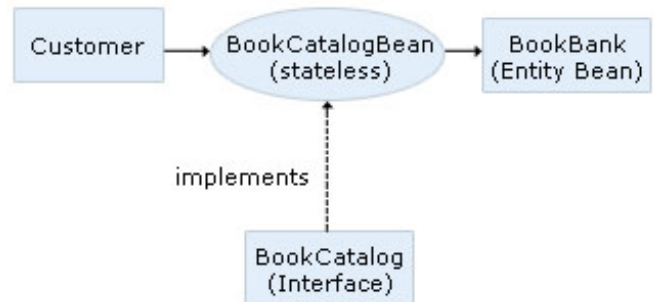
In this part of Enterprise Session Beans, you will learn how to develop, deploy, and run a simple JPA application named **book** using stateless session bean. The purpose of **'book'** is to perform the persistence operations such as Add record and getting information to or from the database.

The 'book' application consists of two enterprise beans, first is **BookBank** that defines the Table name and Primary key in the database, and second one is **BookCatalogBean** that performs the Persistence Object Relational Mapping.

There are following steps that you have to follow to develop a 'book' JEE application.

- 1 Create Remote business interface: **BookCatalogInterface**
- 2 Implement the Annotated Session Bean: **BookCatalogBean**
- 3 Create the Entity bean: **BookBank**
- 4 Create the web client: **WebClient**
- 5 Deploy book on the server.
- 6 Run web client on the web browser.

A UML diagram of this application using JPA can be seen as:



Create Remote Business Interface

To implement a session bean, we first determine the interface that it exposes. In the Book application, this is a simple Java interface declaring all business methods.

package entity.library;

```
import javax.ejb.Remote;
import java.util.Collection;
@Remote
public interface BookCatalogInterface {
    public void addBook(String title, String author, double price);
    public Collection <BookBank>
    getAllBooks();
}
```

II. Implement the annotated session bean

The EJB3 container creates instances of the session bean based on the implementation classes. The application itself never creates session bean instances. It simply asks the

Working with Entity bean using JPA

container for an instance of the session bean to use, either through dependency injection or, for external components, through a JNDI lookup. The class is tagged with the **@Stateless** annotation, which tells the container that the bean object does not maintain any client state information between method invocations. The caller component gets a fresh and random BookCatalogBean instance every time when it makes a bean method call.

In order to use the entity beans in the session bean, you need a special utility class called the EntityManager. The EntityManager acts as a generic DAO (Data Access Object) for all entity beans in the JAR. It translates operations on entity beans to SQL statements to the database. To obtain an EntityManager, the container creates one object and injects it into the session bean.

The addBook() and getAllBooks() methods in the BookCatalogBean class show the EntityManager in action. The EntityManager.persist() method takes a new entity bean POJO and writes it to the database.

The code for the BookCatalogBean is given below.

```
package entity.library;
import java.util.Iterator;
import java.util.Collection;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import java.io.Serializable;
import javax.ejb.Remote;
    @Remote(BookCatalogInterface.class)
    @Stateless
    public class BookCatalogBean implements
    Serializable, BookCatalogInterface {
        @PersistenceContext(unitName="EntityBean")
        EntityManager em;
        protected BookBank book;
        protected Collection <BookBank> bookList;
        public void addBook(String title, String
        author, double price) {
            // Initialize the form
            if (book == null)
                book = new BookBank(title, author, price);
            em.persist(book);
        }
    }
```

```
    }
    public Collection
    <BookBank>getAllBooks() {
        bookList=em.createQuery("from BookBank
        b").getResultList(); return bookList;
    }
}
```

Create BookBank entity bean class:

In the book catalog example, we define a **BookBank** entity bean class. The bean has three properties (title, author and price) to model a Book product. The id property is used to uniquely identify the Book bean instance by the EJB3 container. The id value is automatically generated when the bean is saved to the database.

```
package entity.library;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;
import java.util.Collection;
import javax.persistence.*;
import java.io.Serializable;
@Entity
@Table(name="bookbank")
public class BookBank implements Serializable
{
    long id;
    String title;
    String author;
    double price;
    //protected Collection <LineItems>
    lineitems;
    public BookBank() {
        super();
    }
    public BookBank(String title, String author,
    double price) {
        super();
        this.title = title;
        this.author = author;
        this.price = price;
    }
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    // Getter and setter methods for the
```

Working with Entity bean using JPA

```
defined properties..
public long getId() {
    return id;
}
public void setId(long id) {
    this.id = id;
}
public String getTitle() {
    return title;
}
public void setTitle(String title) {
    this.title = title;
}
public String getAuthor() {
    return author;
}
public void setAuthor(String author) {
    this.author = author;
}
public double getPrice() {
    return price;
}
public void setPrice(double price) {
    this.price = price;
}
}
```

IV. Creating a Web Client

The web client is divided into two pages. First is **"form.jsp"** where a request form is sent to the client; second is **"WebClient.jsp"** which is called from the "form.jsp" page.

The source code for the **"form.jsp"** is given below.

```
<html>
<head>
<title>Library</title>
</head>
<body bgcolor="pink">
<h1>Library</h1>
<hr>
<form action="WebClient.jsp"
method="POST">
<p>Enter the Title:
<input type="text" name="t1"
size="25"></p>
<br>
<p>Enter Author name:
```

```
<input type="text" name="aut" size="25">
</p>
<br>
<p>Enter Price:
<input type="text" name="price" size="25">
</p>
<br>
<p>
<input type="submit" value="Submit">
<input type="reset" value="Reset">
</p>
</form>
</body>
</html>
```

The following statements given below in **"WebClient.jsp"** are used for locating the business interface, creating an enterprise bean instance, and invoking a business method.

```
InitialContext ic = new InitialContext();
BookCatalogInterface bci =
(BookCatalogInterface) ic.lookup("book/
BookCatalogBean/remote");
```

The full source code for the **WebClient.jsp** is given below.

```
<%@ page contentType="text/html;
charset=UTF-8" %>
<%@ page import="entity.library.*,
javax.naming.*, java.util.*"%>
<%!
private BookCatalogInterface bci = null;
String s1,s2,s3;
Collection list;
public void jspInit() {
try {
InitialContext ic = new
InitialContext();
bci = (BookCatalogInterface)
ic.lookup("book/BookCatalogBean/remote");
System.out.println("Loaded Bank Bean");
} catch (Exception ex) {
System.out.println("Error:" +
ex.getMessage());
}
}
public void jspDestroy() {
bci = null;
}
%>
```

Working with Entity bean using JPA

```
<%
try {
    s1 = request.getParameter("t1");
    s2 = request.getParameter("aut");
    s3 = request.getParameter("price");
    if ( s1 != null && s2 != null && s3 !=
null) {
        Double price= new Double(s3);
        bci.addBook(s1, s2, price.doubleValue());
        System.out.println("Record added:");
        %>
        <p>
        <b>Record added</b>
        <p>
        <%
    }
    list=bci.getAllBooks();
    for (Iterator iter = list.iterator());
iter.hasNext();){
        BookBank element =
(BookBank)iter.next();
        %>
        <br>
        <p>Book ID: <b><%= element.getId()
%></b></p>
        <p>Title: <b><%= element.getTitle()
%></b></p>
        <p>Author: <b><%=
element.getAuthor() %></b></p>
        <p>Price: <b><%= element.getPrice() %>
</b></p>
        <%
    }
} // end of try
catch (Exception e) {
    e.printStackTrace ();
}
%>
```

The source code for the "**index.jsp**" is given below that will actual call the client-design form.

```
<%@page language="java" %>
<html>
<head>
<title>Ejb3 JPA Tutorial</title>
</head>
<body bgcolor="#FFFFCC">
<p align="center"><font size="6"
color="#800000"><b>Welcome to <br>
Ejb3-Jboss 4.2.0 Tutorial</b></font>
Click <a href="ejb3/form.jsp">Book Catalog
```

```
Example</a> to execute Library<br></p>
</body>
</html>
```

V. Deploy book application on the Application Server

Before deploying this application, make sure for that you have the following tools on your system.

- **JDK 1.5 or Higher**
- **apache-ant-1.7.0**
- **JBoss 4.2.0**

Do the following steps to deploy the calculator application:

(i) Make a directory structure. You can **Click here** to extract the readymade directory structure according to this tutorial.

(ii) Create the essential deployment descriptor **.xml** files.

build.xml

```
<?xml version="1.0"?>
<project name="Jboss Tutorials" default="all"
basedir=".">
    <target name="init">
        <!-- Define -->
        <property name="dirs.base"
value="\${basedir}"/>
        <property name="classdir"
value="\${dirs.base}/build/classes"/>
        <property name="src"
value="\${dirs.base}/src"/>
        <property name="web"
value="\${dirs.base}/web"/>
        <property
name="deploymentdescription"
value="\${dirs.base}/deploymentdescriptors"/>
    >
        <property name="warFile"
value="book.war"/>
        <property name="earFile"
value="book.ear"/>
        <property name="jarFile"
value="book.jar"/>
        <property name="earDir"
value="\${dirs.base}/build/ear"/>
```

Working with Entity bean using JPA

```
<property name="warDir"
value="\${dirs.base}/build/war"/>
  <property name="jarDir"
value="\${dirs.base}/build/jar"/>
  <!-- classpath for Project -->
  <path id="library.classpath">
    <pathelement path="libext/servlet-
api.jar"/>
    <pathelement path="libext/ejb3-
persistence.jar"/>
    <pathelement path="libext/javaee.jar"/>
  >
  <path id="classpath"/>
  </path>
  <!-- Create Web-inf and classes
directories -->
  <mkdir dir="\${warDir}/WEB-INF"/>
  <mkdir dir="\${warDir}/WEB-INF/
classes"/>
  <!-- Create Meta-inf and classes
directories -->
  <mkdir dir="\${earDir}/META-INF"/>
  <mkdir dir="\${jarDir}/META-INF"/>
  <mkdir dir="\${classdir}"/>
</target>
<!-- Main target -->
<target name="all"
depends="init,build,buildWar,buildJar,buildEar"/>
<!-- Compile Java Files and store in /build/
src directory -->
<target name="build" >
  <javac srcdir="\${src}"
destdir="\${classdir}" debug="true"
includes="**/*.java" >
    <classpath refid="library.classpath"/>
  </javac>
</target>
<!-- Create the web archive File -->
<target name="buildWar" depends="init">
  <copy todir="\${warDir}/WEB-INF/
classes">
    <fileset dir="\${classdir}"
includes="**/*.class" />
  </copy>
  <copy todir="\${warDir}/WEB-INF">
    <fileset
dir="\${deploymentdescription}/web/"
includes="web.xml" />
  </copy>
  <copy todir="\${warDir}">
    <fileset dir="\${web}" includes="**/
```

```
*.*" />
  </copy>
  <!-- Create war file and place in ear
directory -->
  <jar jarfile="\${earDir}/
\${warFile}" basedir="\${warDir}" />
</target>
<!-- Create the jar File -->
<target name="buildJar" depends="init">
  <copy todir="\${jarDir}">
    <fileset dir="\${classdir}"
includes="**/*.class" />
  </copy>
  <copy todir="\${jarDir}/META-INF">
    <fileset
dir="\${deploymentdescription}/jar/"
includes="ejb-jar.xml,weblogic-cmp-rdbms-
jar.xml,weblogic-ejb-jar.xml,persistence.xml"
/ >
  </copy>
  <!-- Create jar file and place in ear
directory -->
  <jar jarfile="\${earDir}/\${jarFile}"
basedir="\${jarDir}" />
</target>
<!-- Create the ear File -->
<target name="buildEar" depends="init">
  <copy todir="\${earDir}/META-INF">
    <fileset
dir="\${deploymentdescription}/ear"
includes="application.xml, jboss-app.xml" />
  </copy>
  <!-- Create ear file and place in ear
directory -->
  <jar jarfile="..\${earFile}"
basedir="\${earDir}" />
  <copy todir="C:/jboss-4.2.0.GA/
server/default/deploy/">
    <fileset dir=".."
includes="\${earFile}" />
  </copy>
</target>
</project>
```

Put this file in the base
(EntityBean\code)directory. Application.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns="http://java.sun.com/
xml/ns/javaee" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
```

Working with Entity bean using JPA

```
version="5" xsi:schemaLocation="http://
java.sun.com/xml/ns/javaee http://
java.sun.com/xml/ns/javaee/
application_5.xsd">
  <display-name>JPA Example</display-
name>
  <module>
    <web>
      <web-uri>book.war</web-uri>
      <context-root>/book</context-root>
    </web>
  </module>
  <module>
    <ejb>book.jar</ejb>
  </module>
</application>
```

jboss-app.xml

The jboss-app.xml file defines a class loader for this application. It makes it simpler for EJB 3.0 to find the default EntityManager.

```
<jboss-app>
  <loader-repository>
    book:archive=book.ear
  </loader-repository>
</jboss-app>
```

Put both files in the **EntityBean\code\deploymentdescriptors\ear** directory.

persistence.xml

The persistence.xml file contains one or several persistence-unit element. Each persistence-unit defines the persistence context name, data source settings, and vendor specific properties. In this example, we are using the HSQL database that is default provided by the Jboss AS. The hibernate property "**create-drop**" will automatically create & drop a table (according to the POJO class) each time when you deploy and run the application on server.

```
<persistence>
  <persistence-unit name="EntityBean">
    <jta-data-source>java:/DefaultDS</jta-
data-source>
    <properties>
      <property
```

```
name="hibernate.hbm2ddl.auto"
      value="create-drop"/>
    <property name="hibernate.dialect"
value="org.hibernate.dialect.HSQLDialect"/>
  </properties>
</persistence-unit>
</persistence>
```

Put this files in the **EntityBean\code\deploymentdescriptors\jar** directory.

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun
Microsystems, Inc.//DTD Web Application 2.3/
/EN" "http://java.sun.com/dtd/web-
app_2_3.dtd">
<web-app >
</web-app>
```

Put this file in the **Stateless\code\deploymentdescriptors\web** directory.

Put all **.jsp** files in the **Stateless\code\web** directory.

Put all **.java** files in the **Stateless\code\src** directory.

(iii) Start command prompt, and go to the **EntityBean\code** directory. Then type the command as:

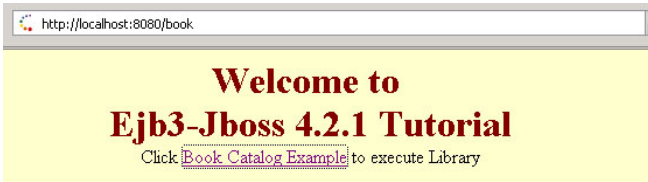
C:\EntityBean\code>ant build.xml

The Ant tool will deploy the **book.ear** file to the **jboss-4.2.0.GA\server\default\deploy** directory.

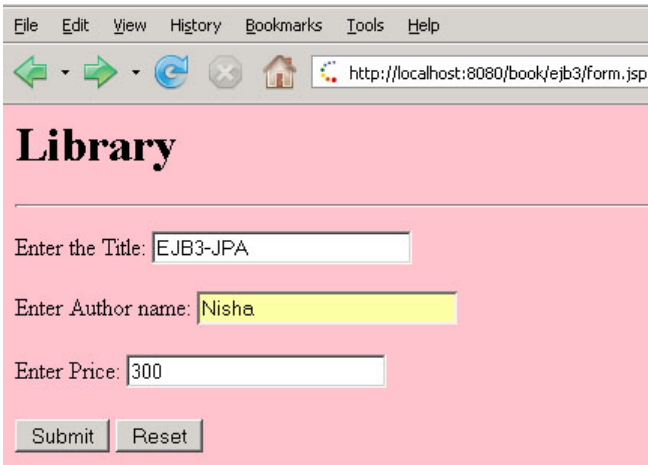
VI. Running the book application
Open the web browser and type the following URL to run the application:

<http://localhost:8080/book>

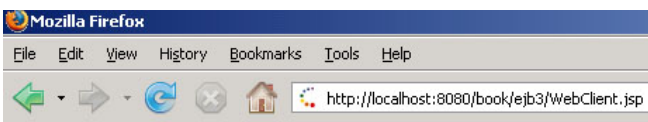
Working with Entity bean using JPA



Click at the given link as **Book Catalog Example**:



Enter the Title, Author and Price for the book to the textbox then clicks the **Submit** button to get the result.



Book ID: **1**
Title: **EJB-JPA**
Author: **Nisha**
Price: **300.00**

Download the full source code



Get
reliable
SERVICES

- Software Solutions
- E-Commerce Solutions
- Website Development
- Web Promotion
- Web Hosting
- Content Development

 **RoseIndia**
<http://www.roseindia.net/services/>

XML Schema

Introduction to XML Schema

XML Schema is a W3C Standard. It is an XML-based alternative to DTDs. It describes the structure of an XML document. The XML Schema language is also referred to as XML Schema Definition (XSD).

In this section, you will learn how to read and create XML Schemas, why XML Schemas are more powerful than DTDs and how to use them in your application. Very soon XML Schemas will be used in most Web applications as a replacement for DTDs. Here are some reasons:

- XML Schemas are extensible to future additions
- XML Schemas are richer and more powerful than DTDs
- XML Schemas are written in XML, supports data types and namespaces.

What is an XML Schema?

XML Schema is used to define the legal building blocks of an XML document, just like a DTD. An XML Schema defines user-defined integrants like elements, sub-elements and attributes needed in an xml document. It defines the data types for elements and attributes along with the occurrence order. It defines whether an element is empty or can include text. It also defines default and fixed values for elements and attributes

Why Use XML Schemas?

XML Schemas are much more powerful than DTDs.

Features of XML Schemas:

XML Schemas Support Data Types

One of the greatest strengths of XML Schemas is its support for data types. With support for data types:

- It is easier to describe allowable document content
- It is easier to validate the correctness of data
- It is easier to work with data from a database
- It is easier to define data facets (restrictions on data)
- It is easier to define data patterns (data formats)
- It is easier to convert data between different data types

XML Schemas use XML Syntax

Another great strength about XML Schemas is that they are written in XML. Simple XML editors are used to edit the Schema files. Even the same XML parsers can be used to parse the Schema files.

XML Schemas are Extensible

XML Schemas are extensible, because they are written in XML. So a user can reuse a Schema in other Schemas and can also refer multiple schemas in the same document. He can also create his own data types derived from the standard types

Well-Formed is not enough alone

A well-formed XML document is a document that conforms to the XML syntax rules. Even if documents are well formed they can still contain errors, and those errors can have serious consequences.

Think of the following situation: you order 5 gross of laser printers, instead of 5 laser printers. With XML Schemas, most of these errors can be caught by your validating software.

XML Schemas Secure Reliable Data Communication

When sending data from a sender to a receiver, it is essential that both parts have the same "expectations" about the content. With XML

XML Schema

Schemas, the sender can describe the data in a way that the receiver will understand. A date like: "03-11-2004" will, in some countries, be interpreted as 3.November and in other countries as 11.March.However, an XML element with a data type like this: `<datatype="date">2004-03-11</date>` ensures a mutual understanding of the content, because the XML data type "date" requires the format "YYYY-MM-DD".

Designing XML Schema

XML documents can have a reference to a DTD or to an XML Schema.

A Simple XML Document

Look at this simple XML document called "**E-mail.xml**":

```
<?xml version="1.0"?>
<E-mail>
<To>Rohan</To>
<From>Amit</From>
<Subject>Surprise....</Subject>
<Body>Be ready for a cruise...</Body>
</E-mail>
```

XML Schema

The following example is a XML Schema file called "**E-mail.xsd**" that defines the elements of the XML document above ("**E-mail.xml**"):

```
<?xml version="1.0"?>

<xs:schema xmlns:xs="http://
www.w3.org/2001/XMLSchema"
targetNamespace="http://
www.roseindia.net"
xmlns="http://www.roseindia.net"
elementFormDefault="qualified">

<xs:element name="E-mail">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="To" type="xs:string"/
>
```

```
    <xs:element name="From"
type="xs:string"/>
    <xs:element name="Subject"
type="xs:string"/>
    <xs:element name="Body"
type="xs:string"/>
  </xs:sequence>
</xs:complexType>
</xs:element>

</xs:schema>
```

We will discuss the building blocks of this schema latter in this section further.

Add a reference to the above-declared XML document

Now this XML document (**E-mail.xml**) has a reference to above-declared XML Schema (**E-mail.xsd**).

```
<?xml version="1.0"?>

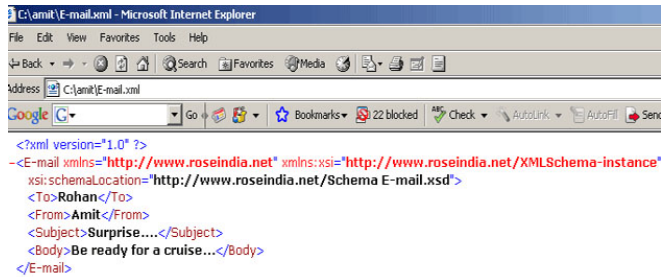
<E-mail
xmlns="http://www.roseindia.net"
xmlns:xsi="http://www.w3.org/2001/
XMLSchema"
xsi:schemaLocation="http://
www.roseindia.net/Schema E-mail.xsd">

  <To>Rohan</To>
  <From>Amit</From>
  <Subject>Surprise....</Subject>
  <Body>Be ready for a cruise...</Body>
</E-mail>
```

In the above xml document xmlns declares the XML namespaces.

Save E-mail.xml and E-mail.xsd in the same location. Open the file E-mail.xml in a web-browser. You will see the following:

XML Schema



Let's briefly discuss the concept of XML Namespaces

XML Namespaces provide a mechanism to avoid element's name conflicts.

Name Conflicts: Since element names in XML are not predefined, chances for frequency to meet name conflict increases when two different documents use the same element names.

We solve the Name Conflicts using a Prefix with a element name: By using a prefix, we can create two different types of elements. Instead of using only prefixes, we add an xmlns attribute to the conflict causing tags to give the prefix a qualified name.

The XML Namespace (xmlns) Attribute: The XML namespace attribute is placed in the start tag of an element and has the following syntax:

```
xmlns:namespace-prefix="namespaceURI"
```

Example 1(taken from E-mail.xml):

```
<E-mail
xmlns="http://www.roseindia.net"
xmlns:xsi="http://www.w3.org/2001/
/XMLSchema"
xsi:schemaLocation="http://www.roseindia.net/
Schema" E-mail.xsd">
```

Example 2(taken from E-mail.xsd):

```
<xs:schema
xmlns:xs="http://www.w3.org/2001/
/XMLSchema"
targetNamespace="http://www.roseindia.net"
xmlns="http://www.roseindia.net"
elementFormDefault="qualified">
```

When a namespace is defined in the start tag of an element, all child elements with the same prefix are associated with the same namespace. In E-mail.xsd "xs" is the defined namespace in the start tag. So it prefixes all the child elements with xs eg...

```
<xs:element name="E-mail">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="To" type="xs:string"/>
      <xs:element name="From" type="xs:string"/>
      <xs:element name="Subject" type="xs:string"/>
    >
    <xs:element name="Body" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
</xs:element>
```

Note that the address used to identify the namespace is not used by the parser to look up information. The only purpose is to give the namespace a unique name. However, very often companies use the namespace as a pointer to a real Web page containing information about the namespace.

Here a Uniform Resource Identifier (URI) is a string of characters, which identifies an Internet Resource.

Default Namespaces: Defining a default namespace for an element saves us from using prefixes in all the child elements. It has the following syntax:

```
xmlns="namespaceURI"
```

We have not included prefixes in all the child element tags (To, From, Subject, Body) in our following example:

```
<?xml version="1.0"?>
<E-mail
xmlns="http://www.roseindia.net"
xmlns:xsi="http://www.w3.org/2001/
/XMLSchema-instance"
xsi:schemaLocation="http://
www.roseindia.net/Schema E-mail.xsd">
<To>Rohan</To>
```

XML Schema

```
<From>Amit</From>
<Subject>Surprise....</Subject>
<Body>Be ready for a cruise...</Body>
</E-mail>
```

Building blocks of a XML-Schema
XSD - The <schema> Element

The <schema> element is the root element of every XML Schema:

```
<?xml version="1.0"?>
<xs:schema>
...
...
</xs:schema>
```

The <schema> element may contain some attributes like...

```
<?xml version="1.0"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.roseindia.net"
xmlns="http://www.roseindia.net"
elementFormDefault="qualified">

...
...

</xs:schema>
```

The following code:

```
xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

indicates that the elements and data types used in the schema come from the "http://www.w3.org/2001/XMLSchema" namespace. It also specifies that the elements and data types that come from the "http://www.w3.org/2001/XMLSchema" namespace should be prefixed with xs:

This code segment

```
targetNamespace="http://www.roseindia.net"
```

indicates that the elements defined by this schema (E-mail, To, From, Subject, Body.) come

from the "http://www.roseindia.net" namespace.

This fragment:

```
xmlns="http://www.roseindia.net"
```

indicates that the default namespace is "http://www.roseindia.net".

This fragment:

```
elementFormDefault="qualified"
```

indicates that any elements used by the XML instance document which were declared in this schema must be a namespace qualified.

Referencing a Schema in an XML Document

This XML document (E-mail.xml) has a reference to an XML Schema (E-mail.xsd).

```
<?xml version="1.0"?>

<E-mail
xmlns="http://www.roseindia.net"
xmlns:xsi="http://www.w3.org/2001/XMLSchema"
xsi:schemaLocation="http://www.roseindia.net/Schema E-mail.xsd">

<To>Rohan</To>
<From>Amit</From>
<Subject>Surprise....</Subject>
<Body>Be ready for a cruise...</Body>
</E-mail>
```

The following fragment:

```
xmlns="http://www.roseindia.net"
```

Specifies the default namespace declaration. This declaration tells the schema-validator that all the elements used in this XML document are declared in the "http://www.w3schools.com" namespace.

Once you have the XML Schema Instance namespace available:

```
xmlns:xsi="http://www.w3.org/2001/
```

XML Schema

XMLSchema-instance"

you can use the schemaLocation attribute. This attribute has two values. The first value is the namespace to use. The second value is the location of the XML schema to use for that namespace:

```
xsi:schemaLocation="http://www.roseindia.net
note.xsd"
```

XSD Simple Elements

XML Schemas define the elements of XML files.

XML simple element contains only text not even any other elements or attributes. But the text can be of many different types. It can be among the types included in the XML Schema definition (boolean, string, date, etc.), or it may be a custom type that a user is free to define. Even. Restrictions (facets) can be added to a data type in order to limit its content.

Defining a Simple Element

The syntax for a simple element is:

```
<xs:element name="aaa" type="bbb"/>
```

where **aaa** is the name of the element and **bbb** is the data type of the element.

XML Schema has a lot of built-in data types.

The most common types are:

- xs:string
- xs:decimal
- xs:integer
- xs:Boolean
- xs:date
- xs:time

Example:

Few of XML elements:

```
<name>Rahul</name>
<age>15</age>
<currentdate>2007-05-15</currentdate>
```

The corresponding simple element definitions:

```
<xs:element name="name" type="xs:string"/>
<xs:element name="age" type="xs:integer"/>
<xs:element name="currentdate"
type="xs:date"/>
```

Default Values for Simple Elements

Simple elements may have a specified default value OR a fixed specified value. A default value is automatically assigned to the element when no other value is specified for example to set the "orange" default value.

```
<xs:element name="fruit" type="xs:string"
default="orange"/>
```

Fixed Values for Simple Elements

A fixed value is also automatically assigned to the element, and it cannot further specify another value.

In the following example the fixed value is "apple":

```
<xs:element name="fruit" type="xs:string"
fixed="apple"/>
```

XSD Complex Elements:

A complex element contains other elements or attributes.

What is a Complex Element?

It is an XML element that contains other elements and/or attributes. They are of four types:

- empty elements
- elements that contain only other elements
- elements that contain only text
- elements that contain both other elements and text

XML Schema

Note: Each of these elements may contain attributes as well!

Examples of Complex Elements

A complex empty XML element, "employee"

```
<employee eid="1234"/>
```

A complex XML element, "employee", which contains only other elements:

```
<employee>
<firstname>Amit</firstname>
<lastname>Gupta</lastname>
</employee>
```

A complex XML element, "employee", which contains only text:

```
<employee type="category">Programmer</
employee>
```

A complex XML element, "event", which contains both elements and text:

```
<event>
It occurred on <date
lang="norwegian">15.05.07</date> ....
</event>
```

Defining a Complex Element:

Look at this complex XML element, "employee", which contains only other elements:

```
<employee>
<firstname>Amit</firstname>
<lastname>Gupta</lastname>
</employee>
```

We can define a complex element in an XML Schema in two different ways:

1. "employee" element can be declared directly by naming the element, like this:

```
<xs:element name="employee">
<xs:complexType>
<xs:sequence>
```

```
<xs:element name="firstname"
type="xs:string"/>
<xs:element name="lastname"
type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
```

In the above described method only "employee" element can use the specified complex type. Note that the child elements, "firstname" and "lastname", are surrounded by the <sequence> indicator. This means that the child elements must appear in the same order as they are declared.

2. "employee" element can have a type attribute referring to the name of the complex type to use:

```
<xs:element name="employee"
type="personinfo"/>
<xs:complexType name="personinfo">
<xs:sequence>
<xs:element name="firstname"
type="xs:string"/>
<xs:element name="lastname"
type="xs:string"/>
</xs:sequence>
</xs:complexType>
```

Using the method described above, several elements can refer to the same complex type, like this:

```
<xs:element name="employee"
type="personinfo"/>
<xs:element name="employer"
type="personinfo"/>
<xs:element name="teammember"
type="personinfo"/>
<xs:complexType name="personinfo">
<xs:sequence>
<xs:element name="firstname"
type="xs:string"/>
<xs:element name="lastname"
type="xs:string"/>
</xs:sequence>
</xs:complexType>
```

XML Schema

XSD Attributes

All attributes are declared as simple types.

What is an Attribute?

Simple elements do not contain attributes. If an element has attributes, then it is of a complex type element. But the attribute itself is always declared as a simple type.

Defining an Attribute?

The syntax for defining an attribute is:

```
<xs:attribute name="aaa" type="bbb"/>
```

where **aaa** is the name of the attribute and **bbb** specifies the data type of the attribute.

XML Schema has a lot of built-in data types.

The most common types are:

- xs:string
- xs:decimal
- xs:integer
- xs:Boolean
- xs:date
- xs:time

Example:

Here is an XML element with an attribute:

```
<name lang="EN">Rahul</name>
```

And here is the corresponding attribute definition:

```
<xs:attribute name="lang" type="xs:string"/>
```

Default Values for Attributes

Attributes may have a specified default value OR a specified fixed value. A default value is automatically assigned to the attribute when no other value is specified for example..the default value is "EN":

```
<xs:attribute name="lang" type="xs:string" default="EN"/>
```

Fixed Values for Attributes

A fixed value is automatically assigned to the attribute, and it cannot further specify another value for example. The fixed value is "EN":

```
<xs:attribute name="lang" type="xs:string" fixed="EN"/>
```

Optional and Required Attributes

Attributes are optional by default. To specify that the attribute is required, use the "use" attribute:

```
<xs:attribute name="lang" type="xs:string" use="required"/>
```

Restrictions on Content

When an XML element or attribute has a data type defined, it can put restrictions on the element's or attribute's content. If an XML element is of type "xs:age" and contains a string like "Hello", the element will not validate. With XML Schemas, user can also add his own restrictions to XML elements and attributes. These restrictions are called **facets**.

XML- SAX Parser using JAXP API

XML- SAX Parser using JAXP API

In the previous issue of Java Jazz Up you have read about XML technology. In this issue, you will learn how XML technology works with Java using different kinds of XML parsers. Lets quickly focus on the overview of XML.

XML is a W3C Recommendations. It stands for Extensible Markup Language. It is a markup language much like HTML used to describe data. In XML, tags are not predefined. A user defines his own tags and XML document structure like Document Type Definition (DTD), XML Schema to describe the data. Hence it is self-descriptive too. There is nothing special about XML It is just plain text with the addition of some XML tags enclosed in angle brackets. In a simple text editor, the XML document is easily visible.

Below is a list of XML-related technologies:

Technology	Document.
DTD (Document Type Definition)	It is used to define the legal elements in an XML document.
XSD (XML Schema)	It is an XML-based alternative to DTDs.
XHTML (Extensible HTML)	It is a stricter and cleaner version of HTML.
XSL (Extensible Style Sheet Language)	XSL consists of three parts: XSLT - a language for transforming XML documents, XPath - a language for navigating in XML documents, and XSL-FO - a language for formatting XML documents.
XSLT (XSL Transformations)	It is used to transform XML documents into other XML formats, like XHTML.
XML DOM (XML Document Model)	Defines a standard way for accessing and Object manipulating XML documents.
XPath	It is a language for navigating in XML documents.
XSL-FO (Extensible Sheet Language Formatting Objects)	It is an XML based markup language describing the Style formatting of XML data for output to screen, paper or other media.
XLink (XML Linking Language)	It is a language for creating hyperlinks in XML documents
XPointer (XML Pointer Language)	It allows the XLink hyperlinks to point to more specific parts in the XML document.
XForms (XML Forms)	It uses XML to define form data.
XQuery (XML Query Language)	It is designed to query XML data.
SOAP (Simple Object Access Protocol)	It is an XML-based protocol to let applications exchange information over HTTP.
WSDL (Web Services Description Language)	It is an XML-based language for describing web services.
RDF(Resource Description Framework)	It is an XML-based language for describing web resources.
RSS (Really Simple Syndication)	It is a format for syndicating news and the content of news-like sites.
WAP(Wireless Application Protocol)	It was designed to show internet contents on wireless clients, like mobile phones.
SMIL (Synchronized Multimedia Integration Language)	It is a language for describing audiovisual presentations.

XML- SAX Parser using JAXP API

SVG (Scalable Vector Graphics) Defines graphics in XML format.

Introduction to XML Parser:

In computing terms, a parser is a program that takes input in the form of sequential instructions, tags, or some other defined sequence of tokens, and breaks them up into easily manageable parts.

XML parser is used to read, update, create and manipulate an XML document. Whenever the XML document executes, the parser recognizes and responds to each XML structure taking some specified action based on the structure type.

XML parsers can be validating or nonvalidating. Validating parser checks the contents of a document against a set of specific rules i.e. in what order they must appear. These rules appear in an XML document either as an optional XML structure called a document type definition, or DTD, or as an XML Schema. Nonvalidating parsers are smaller and faster, but they do not check documents against the DTD. They only check whether the XML document is structurally well formed or not.

Parsing XML Documents

To manipulate an XML document, XML parser is needed. The parser loads the document into the computer's memory. Once the document is loaded, its data can be manipulated using the appropriate parser.

We will soon discuss APIs and parsers for accessing XML documents using serially access mode (SAX) and random access mode (DOM). The specifications to ensure the validity of XML documents are DTDs and the Schemas.

DOM (Document Object Model)

The XML Document Object Model (XML DOM) defines a standard way to access and manipulate XML documents using any programming language (and a parser for that language).

The DOM presents an XML document as a tree-

structure (a node tree), with the elements, attributes, and text defined as nodes. DOM provides access to the information stored in your XML document as a hierarchical object model.

The DOM converts an XML document into a collection of objects in an object model in a tree structure (which can be manipulated in any way). The textual information in XML document gets turned into a bunch of tree nodes and a user can easily traverse through any part of the object tree, any time. This makes easier to modify the data, to remove it, or even to insert a new one. This mechanism is also known as the random access protocol.

DOM is very useful when the document is small. DOM reads the entire XML structure and holds the object tree in memory, so it is much more CPU and memory intensive. The DOM is most suited for interactive applications because the entire object model is present in memory, where it can be accessed and manipulated by the user.

SAX (Simple API for XML)

This API was an innovation, made on the XML-DEV mailing list through product collaboration, rather than being a product of the W3C.

SAX (Simple API for XML) like DOM gives access to the information stored in XML documents using any programming language (and a parser for that language).

This standard API works in serial access mode to parse XML documents. This is a very fast-to-execute mechanism employed to read and write XML data comparing to its competitors. SAX tells the application, what is in the document by notifying through a stream of parsing events. Application then processes those events to act on data.

SAX is also called as an event-driven protocol, because it implements the technique to register the handler to invoke the callback methods whenever an event is generated. Event is generated when the parser encounters a new XML tag or encounters an error, or wants to tell anything else. SAX is memory-efficient to a

XML- SAX Parser using JAXP API

great extend.

SAX is very useful when the document is large. DOM reads the entire XML structure and holds the object tree in memory, so it is much more CPU and memory intensive. For that reason, the SAX API are preferred for server-side applications and data filters that do not require any memory intensive representation of the data.

Overview of JAXP API's

JAXP (Java API for XML Processing) doesn't do any kind of processing instead it provides a mechanism to obtain parsed XML documents employing SAX and DOM parsers. JAXP provides a mechanism to plug-in with various providers (supporting standard specifications for DOM, SAX and XSLT). JAXP also specifies which provider to use.

Overview of the main JAXP API Packages

The libraries that define needed JAXP APIs are:

JAXP APIs	Description
javax.xml.parsers	The JAXP APIs provide a common interface for different vendors' to use SAX and DOM parsers.
org.w3c.dom	Defines the Document class (a DOM) along with the classes for all of the components of a DOM.
org.xml.sax	Defines the basic SAX APIs.

javax.xml.transform Defines the XSLT APIs that let's to transform XML into other forms.

The SAX API is defined in org.xml.sax package of JAXP-APIs. The "Simple API" for XML (SAX) is the event-driven, serial-access mechanism that does element-by-element processing. The API for this level reads and writes XML to a data repository or the Web.

The DOM API is defined in org.w3c.dom package of JAXP-APIs. The DOM API is easier to use. It provides a tree structure of objects. The DOM API is used to manipulate the hierarchy of application objects it encapsulates.

The XSLT APIs defined in javax.xml.transform package of JAXP-APIs. The XSLT APIs let you convert XML data to into other forms.

javax.xml.parsers —Description

Provides classes to process XML documents and supports two types of pluggable parsers ie. SAX and DOM Here are the following classes defined in javax.xml.parsers package:

Classes	Description
DocumentBuilder	Defines the API to obtain DOM Document instances from an XML document.
DocumentBuilderFactory	Defines a factory API that enables applications to obtain a parser that produces DOM object trees from XML documents
SAXParser	Defines the API that wraps an XMLReader implementation class
SAXParserFactory	Defines a factory API that enables document.applications to configure and obtain a SAX based parser to parse XML documents

This package contains two vendor-neutral factory classes: **SAXParserFactory** (builds a SAXParser) and **DocumentBuilderFactory**

XML- SAX Parser using JAXP API

(builds a DocumentBuilder).

The DocumentBuilder further creates a DOM-compliant document object.

The factory APIs enables to plug-in with the XML implementation (provided by any vendor without changing the source code). The obtained implementation depends on the setting of the system properties of these factory classes `javax.xml.parsers.SAXParserFactory` and `javax.xml.parsers.DocumentBuilderFactory`. The default values (unless overridden at runtime) point to the reference implementation.

In this section, you will learn how **SAX Parser** parses and get the information from the XML document.

The SAX Packages:

The SAX parser is defined in the following packages:

Package	Description
<code>org.xml.sax</code>	Defines the SAX interfaces. The name "org.xml" is the package prefix that was settled on by the group that defined the SAX API.
<code>org.xml.sax.ext</code>	Defines SAX extensions that are used when doing more sophisticated SAX processing, for example, to process a document type definitions (DTD) or to see the detailed syntax for a file.
<code>org.xml.sax.helpers</code>	Contains helper classes that make it easier to use SAX — for example, by

defining a default handler that has null-methods for all of the interfaces, so you only need to override the ones you actually want to implement.

<code>javax.xml.parsers</code>	Defines the <code>SAXParserFactory</code> class, which returns the
<code>SAXParser</code> . Also	defines exception classes for reporting errors.

Main classes of javax.xml.parsers package:

<code>SAXParser</code>	Defines the API that wraps an <code>XMLReader</code> implementation class
<code>SAXParserFactory</code>	Defines a factory API that enables applications to configure and obtain a SAX based parser to parse XML documents

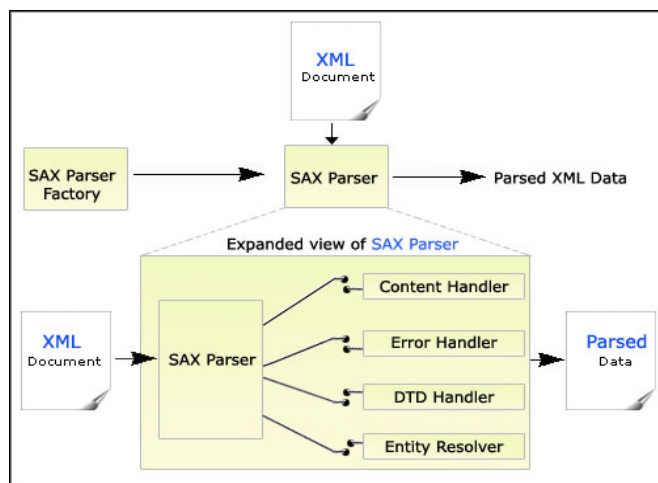
Main interfaces of org.xml.sax package:

<code>ContentHandler</code>	Receive notification of the logical content of a document.
<code>DTDHandler</code>	Receive notification of basic DTD-related events.
<code>EntityResolver</code>	Basic interface for resolving entities.
<code>ErrorHandler</code>	Basic interface for SAX error handlers.

XML- SAX Parser using JAXP API

Understanding SAX Parser

At the very first, create an instance of the **SAXParserFactory** class, which generates an instance of the parser. This parser wraps a SAXReader object. When the parser's parse() method is invoked, the reader invokes one of the several callback methods (implemented in the application). These callback methods are defined by the interfaces ContentHandler, ErrorHandler, DTDHandler, and EntityResolver.



Brief description of the key SAX APIs:

SAXParserFactory:

SAXParserFactory object creates an instance of the parser determined by the system property, using the class `javax.xml.parsers.SAXParserFactory`.

SAXParser:

The SAXParser interface defines several kinds of parse() methods. Generally, XML data source and a DefaultHandler object is passed to the parser. This parser processes the XML file and invokes the appropriate method on the handler object.

SAXReader:

The SAXParser wraps a SAXReader (may use

SAXParser's getXMLReader() and configure it). It is the SAXReader, which carries on the conversation with the SAX event handlers you define.

DefaultHandler:

Not shown in the diagram, a DefaultHandler implements the ContentHandler, ErrorHandler, DTDHandler, and EntityResolver interfaces (with null methods). You override only the ones you're interested in.

ContentHandler:

Methods like startDocument, endDocument, startElement, and endElement are invoked when an XML tag is recognized. This interface also defines methods characters and processingInstruction, which are invoked when the parser encounters the text in an XML element or an inline processing instruction, respectively.

ErrorHandler:

Methods error, fatalError, and warning are invoked in response to various parsing errors. The default error handler throws an exception for fatal errors and ignores other errors (including validation errors). To ensure the correct handling, you'll need to supply your own error handler to the parser.

DTDHandler:

Defines methods you will rarely call. Used while processing a DTD to recognize and act on declarations for an unparsed entity.

EntityResolver:

The resolveEntity method is invoked when the parser needs to identify the data referenced by a URI. .

Lets see an example of parsing an XML document using Java SAX Parser. In this section, we are going to develop a simple Java program named **EmpSaxParser.java** that first determines whether a XML document is well-formed or not then retrieves data from XML

XML- SAX Parser using JAXP API

document using JAXP APIs.

Description of the program:

In this example you need a well-formed XML file that has some data (Emp_Id, Emp_Name and Emp_E-mail in our case).

Here is the XML File to be parsed: **Employee-Detail.xml**

```
<?xml version = "1.0" ?>
<Employee-Detail>
<Employee>
<Emp_Id> E-001 </Emp_Id>
<Emp_Name> Vinod </Emp_Name>
<Emp_E-mail> Vinod1@yahoo.com </
Emp_E-mail>
</Employee>
<Employee>
<Emp_Id> E-002 </Emp_Id>
<Emp_Name> Amit </Emp_Name>
<Emp_E-mail> Amit2@yahoo.com </Emp_E-
mail>
</Employee>
<Employee>
<Emp_Id> E-003 </Emp_Id>
<Emp_Name> Deepak </Emp_Name>
<Emp_E-mail> Deepak3@yahoo.com </
Emp_E-mail>
</Employee>
</Employee-Detail>
```

Develop a java file (**EmpSaxParser.java**) that uses an xml file to parse and check its well-formed ness. Initially the program checks that the given file exists or not by using **exists()** method. Here is a sample code of this program:

```
File file = new File(str);
if (file.exists()){
XMLReader reader =
XMLReaderFactory.createXMLReader();
reader.parse(str);
System.out.println(str + " is well-
formed ");
```

The **XMLReaderFactory** helps in creating an XML reader, which parses xml document using the appropriate callbacks. And it determines that the parsed xml is well-formed or not. If xml document is will-formed, it will display a message "Employee-Detail.xml is well-formed!" Otherwise

prints "Employee-Detail.xml isn't well-formed!". If you enter a file that doesn't exist it will show "File not found!".

Now, lets see the sample code to parse the data:

```
public void startElement(String uri,
String localName,
String element_name, Attributes
attributes)throws SAXException{
if (element_name.equals("Emp_Id")){
id = true;
}
if
(element_name.equals("Emp_Name")){
name = true;
}
if (element_name.equals("Emp_E-
mail")){
mail = true;
}
}

public void characters(char[] ch, int start, int
len) throws SAXException{
String str = new String (ch, start,
len);
if (id){
System.out.println("Emp_Id:
"+str);
id = false;
}
if (name){
System.out.println("Name: "+str);
name = false;
}
if (mail){
System.out.println("E-mail: "+str);
mail = false;
}
}
};
```

parser.parse(str, dHandler);

If the given file exists and well-formed then the instance of **SAXParser** class parses the file using the **parse()** method. As long as the **startElement()** method returns 'true', the **characters()** method prints data .

XML- SAX Parser using JAXP API

Characters(char[] ch, int start, int len)

method retrieves identification of character data. The Parser calls this method and to report every character data encountered. If any error occurs it throws the **SAXException**. This method takes the following parameters:

ch: This is the characters of XML document.

start: This is starting position in an array.

len: This is the number of characters to read from an array.

Here is full source code of

EmpSaxParser.java:

```
import javax.xml.parsers.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import java.io.*;
public class EmpSaxParser{
    public static void main(String[] args) throws
    IOException{
        BufferedReader bf = new
        BufferedReader(new
        InputStreamReader(System.in));
        System.out.print("Enter XML file name:");
        String xmlFile = bf.readLine();
        EmpSaxParser detail = new
        EmpSaxParser(xmlFile);
    }
    public EmpSaxParser(String str){
        try{
            File file = new File(str);
            if (file.exists()){
                XMLReader reader =
                XMLReaderFactory.createXMLReader();
                reader.parse(str);
                System.out.println(str + " is well-
                formed ");
                System.out.println("");
                SAXParserFactory parserFact =
                SAXParserFactory.newInstance();
                SAXParser parser =
                parserFact.newSAXParser();
                System.out.println("XML Data: ");
                DefaultHandler dHandler = new
                DefaultHandler(){
                    boolean id;
                    boolean name;
                    boolean mail;
                };
                parser.setContentHandler(dHandler);
                parser.parse(str);
            }
            else{
                System.out.println("File not found!");
            }
        }
        catch (SAXException sax){
            System.out.println(str + " isn't well-
            formed");
        }
        catch (Exception e){
            System.out.println("XML File hasn't any
            elements");
            e.printStackTrace();
        }
    }
}
```

```
public void startElement(String uri,
String localName, String element_name,
Attributes attributes)throws SAXException{
    if (element_name.equals("Emp_Id")){
        id = true;
    }
    if
    (element_name.equals("Emp_Name")){
        name = true;
    }
    if (element_name.equals("Emp_E-
    mail")){
        mail = true;
    }
}
public void characters(char[] ch, int
start, int len) throws SAXException{
    String str = new String (ch, start,
    len);
    if (id){
        System.out.println("Emp_Id:
    "+str);
        id = false;
    }
    if (name){
        System.out.println("Name: "+str);
        name = false;
    }
    if (mail){
        System.out.println("E-mail: "+str);
        mail = false;
    }
}
};
parser.parse(str, dHandler);
}
else{
    System.out.println("File not found!");
}
}
catch (SAXException sax){
    System.out.println(str + " isn't well-
    formed");
}
catch (Exception e){
    System.out.println("XML File hasn't any
    elements");
    e.printStackTrace();
}
}
```

XML- SAX Parser using JAXP API

Output of the Program:

```
C:\xml>javac EmpSaxParser.java
```

```
C:\xml>java EmpSaxParser
Enter XML file name:emp.xml
emp.xml is well-formed
```

XML Data:

Emp_Id: E-001

Name: Vinod

E-mail: Vinod1@yahoo.com

Emp_Id: E-002

Name: Amit

E-mail: Amit2@yahoo.com

Emp_Id: E-003

Name: Deepak

E-mail: Deepak3@yahoo.com

**Coffee with
JavaFX**

**Enjoy
A
New
Flavour**



Visit us

<http://www.roseindia.net>

Hibernate Query Language

Introduction to Hibernate Query Language

Hibernate Query Language or HQL for short is extremely powerful query language. HQL is much like SQL and are case-insensitive, except for the names of the Java Classes and properties. Hibernate Query Language is used to execute queries against database. Hibernate automatically generates the sql query and execute it against underlying database if HQL is used in the application. HQL is based on the relational object models and makes the SQL object oriented. Hibernate Query Language uses Classes and properties instead of tables and columns. Hibernate Query Language is extremely powerful and it supports Polymorphism, Associations, Much less verbose than SQL.

There are other options that can be used while using Hibernate. These are Query By Criteria (QBC) and Query BY Example (QBE) using Criteria API and the Native SQL queries. Features HQL:

- **Full support for relational operations:** HQL allows representing SQL queries in the form of objects. Hibernate Query Language uses Classes and properties instead of tables and columns.
- **Return result as Object:** The HQL queries return the query result(s) in the form of object(s), which is easy to use. These eliminate the need of creating the object and populate the data from result set.
- **Polymorphic Queries:** HQL fully supports polymorphic queries. Polymorphic queries results the query results along with all the child objects if any.
- **Easy to Learn:** Hibernate Queries are easy to learn and it can be easily implemented in the applications.
- **Support for Advance features:** HQL contains many advance features such

as pagination, fetch join with dynamic profiling, Inner/outer/full joins, Cartesian products. It also supports Projection, Aggregation (max, avg) and grouping, Ordering, Sub queries and SQL function calls.

- **Database independent:** Queries written in HQL are database independent (If database supports the underlying feature).

Understanding HQL Syntax

Any Hibernate Query Language may consist of following elements:

- Clauses
- Aggregate functions
- Subqueries

Clauses in the HQL are:

- from
- select
- where
- order by
- group by

Aggregate functions are:

- avg(...), sum(...), min(...), max(...)
- count(*)
- count(...), count(distinct ...), count(all...)

Subqueries:

Subqueries are nothing but a query within another query. Hibernate supports Subqueries if the underlying database supports it.

Lets understand, how these HQL queries are implemented in a POJO based class.

Preparing table for HQL Examples

Create **insurance** table and populate it with the data. To create the insurance table and insert the sample data, run the following sql query:

Hibernate Query Language

```
/*Table structure for table `insurance` */

drop table if exists `insurance`;

CREATE TABLE `insurance` (
  `ID` int(11) NOT NULL default '0',
  `insurance_name` varchar(50) default NULL,
  `invested_amount` int(11) default NULL,
  `investment_date` datetime default NULL,
  PRIMARY KEY (`ID`)
) TYPE=MyISAM;

/*Data for the table `insurance` */

insert into `insurance` values (1,'Car
Insurance',1000,'2005-01-05 00:00:00');
insert into `insurance` values (2,'Life
Insurance',100,'2005-10-01 00:00:00');
insert into `insurance` values (3,'Life
Insurance',500,'2005-10-15 00:00:00');
insert into `insurance` values (4,'Car
Insurance',2500,'2005-01-01 00:00:00');
insert into `insurance` values (5,'Dental
Insurance',500,'2004-01-01 00:00:00');
insert into `insurance` values (6,'Life
Insurance',900,'2003-01-01 00:00:00');
insert into `insurance` values (7,'Travel
Insurance',2000,'2005-02-02 00:00:00');
insert into `insurance` values (8,'Travel
Insurance',600,'2005-03-03 00:00:00');
insert into `insurance` values (9,'Medical
Insurance',700,'2005-04-04 00:00:00');
insert into `insurance` values (10,'Medical
Insurance',900,'2005-03-03 00:00:00');
insert into `insurance` values (11,'Home
Insurance',800,'2005-02-02 00:00:00');
insert into `insurance` values (12,'Home
Insurance',750,'2004-09-09 00:00:00');
insert into `insurance` values (13,'Motorcycle
Insurance',900,'2004-06-06 00:00:00');
insert into `insurance` values (14,'Motorcycle
Insurance',780,'2005-03-03 00:00:00');
```

Create POJO class for Insurance table:

Here is the code of our java file (Insurance.java), which will map the POJO objects to the insurance table.

```
package roseindia.tutorial.hibernate;
```

```
import java.util.Date;

public class Insurance {
  private long lngInsuranceId;
  private String insuranceName;
  private int investementAmount;
  private Date investementDate;

  /**
   * @return Returns the insuranceName.
   */
  public String getInsuranceName() {
    return insuranceName;
  }
  /**
   * @param insuranceName The
   insuranceName to set.
   */
  public void setInsuranceName(String
insuranceName) {
    this.insuranceName = insuranceName;
  }
  /**
   * @return Returns the
   investementAmount.
   */
  public int getInvestementAmount() {
    return investementAmount;
  }
  /**
   * @param investementAmount The
   investementAmount to set.
   */
  public void setInvestementAmount(int
investementAmount) {
    this.investementAmount =
investementAmount;
  }
  /**
   * @return Returns the investementDate.
   */
  public Date getInvestementDate() {
    return investementDate;
  }
  /**
   * @param investementDate The
   investementDate to set.
   */
  public void setInvestementDate(Date
investementDate) {
    this.investementDate = investementDate;
  }
}
```


Hibernate Query Language

```
}
/**
 * @return Returns the lngInsuranceId.
 */
public long getLngInsuranceId() {
    return lngInsuranceId;
}
/**
 * @param lngInsuranceId The
lngInsuranceId to set.
 */
public void setLngInsuranceId(long
lngInsuranceId) {
    this.lngInsuranceId = lngInsuranceId;
}
}
```

Adding mappings to contact.hbm.xml file:

```
<class
name="roseindia.tutorial.hibernate.Insurance"
table="insurance">
    <id name="lngInsuranceId" type="long"
column="ID" >
        <generator class="increment"/>
    </id>

    <property name="insuranceName">
        <column name="insurance_name" />
    </property>
    <property name="investmentAmount">
        <column name="invested_amount" />
    </property>
    <property name="investmentDate">
        <column name="investment_date" />
    </property>
</class>
```

Now here is the simplest code of **SelectHQLExample.java** that is using **from** clause to get all records from the insurance table.

package roseindia.tutorial.hibernate;

```
import org.hibernate.Session;
import org.hibernate.*;
import org.hibernate.cfg.*;

import java.util.*;

public class SelectHQLExample {
```

```
    public static void main(String[] args) {
        Session session = null;

        try{
            // This step will read hibernate.cfg.xml and
            prepare hibernate for use
            SessionFactory sessionFactory = new
            Configuration().configure().buildSessionFactory();
            session =sessionFactory.openSession();

            //Using from Clause
            String SQL_QUERY ="from Insurance
insurance";
            Query query =
            session.createQuery(SQL_QUERY);
            for(Iterator
            it=query.iterate();it.hasNext();){
                Insurance
                insurance=(Insurance)it.next();
                System.out.println("ID: " +
                insurance.getLngInsuranceId());
                System.out.println("First Name: " +
                insurance.getInsuranceName());
            }
            session.close();
        }catch(Exception e){
            System.out.println(e.getMessage());
        }finally{
        }
    }
}
```

To run the example select Run-> Run As -> Java Application from the menu bar. Following out is displayed in the Eclipse console:

```
log4j:WARN No appenders could be found for
logger (org.hibernate.cfg.Environment).
log4j:WARN Please initialize the log4j system
properly.
Hibernate: select insurance0_.ID as col_0_0_
from insurance insurance0_
ID: 1
Hibernate: select insurance0_.ID as ID0_,
insurance0_.insurance_name as
insurance2_2_0_,
insurance0_.invested_amount as
invested3_2_0_,
insurance0_.investment_date as
investem4_2_0_ from insurance insurance0_
where insurance0_.ID=?
First Name: Car Insurance
```

Hibernate Query Language

ID: 2

```
Hibernate: select insurance0_.ID as ID0_,
insurance0_.insurance_name as
insurance2_2_0_,
insurance0_.invested_amount as
invested3_2_0_,
insurance0_.investment_date as
investem4_2_0_ from insurance insurance0_
where insurance0_.ID=?
```

First Name: Life Insurance

ID: 3

```
Hibernate: select insurance0_.ID as ID0_,
insurance0_.insurance_name as
insurance2_2_0_,
insurance0_.invested_amount as
invested3_2_0_,
insurance0_.investment_date as
investem4_2_0_ from insurance insurance0_
where insurance0_.ID=?
```

First Name: Life Insurance

ID: 4

```
Hibernate: select insurance0_.ID as ID0_,
insurance0_.insurance_name as
insurance2_2_0_,
insurance0_.invested_amount as
invested3_2_0_,
insurance0_.investment_date as
investem4_2_0_ from insurance insurance0_
where insurance0_.ID=?
```

First Name: Car Insurance

ID: 5

```
Hibernate: select insurance0_.ID as ID0_,
insurance0_.insurance_name as
insurance2_2_0_,
insurance0_.invested_amount as
invested3_2_0_,
insurance0_.investment_date as
investem4_2_0_ from insurance insurance0_
where insurance0_.ID=?
```

First Name: Dental Insurance

ID: 6

```
Hibernate: select insurance0_.ID as ID0_,
insurance0_.insurance_name as
insurance2_2_0_,
insurance0_.invested_amount as
invested3_2_0_,
insurance0_.investment_date as
investem4_2_0_ from insurance insurance0_
where insurance0_.ID=?
```

First Name: Life Insurance

ID: 7

```
Hibernate: select insurance0_.ID as ID0_,
```

```
insurance0_.insurance_name as
insurance2_2_0_,
insurance0_.invested_amount as
invested3_2_0_,
insurance0_.investment_date as
investem4_2_0_ from insurance insurance0_
where insurance0_.ID=?
```

First Name: Travel Insurance

ID: 8

```
Hibernate: select insurance0_.ID as ID0_,
insurance0_.insurance_name as
insurance2_2_0_,
insurance0_.invested_amount as
invested3_2_0_,
insurance0_.investment_date as
investem4_2_0_ from insurance insurance0_
where insurance0_.ID=?
```

First Name: Travel Insurance

ID: 9

```
Hibernate: select insurance0_.ID as ID0_,
insurance0_.insurance_name as
insurance2_2_0_,
insurance0_.invested_amount as
invested3_2_0_,
insurance0_.investment_date as
investem4_2_0_ from insurance insurance0_
where insurance0_.ID=?
```

First Name: Medical Insurance

ID: 10

```
Hibernate: select insurance0_.ID as ID0_,
insurance0_.insurance_name as
insurance2_2_0_,
insurance0_.invested_amount as
invested3_2_0_,
insurance0_.investment_date as
investem4_2_0_ from insurance insurance0_
where insurance0_.ID=?
```

First Name: Medical Insurance

ID: 11

```
Hibernate: select insurance0_.ID as ID0_,
insurance0_.insurance_name as
insurance2_2_0_,
insurance0_.invested_amount as
invested3_2_0_,
insurance0_.investment_date as
investem4_2_0_ from insurance insurance0_
where insurance0_.ID=?
```

First Name: Home Insurance

ID: 12

```
Hibernate: select insurance0_.ID as ID0_,
insurance0_.insurance_name as
insurance2_2_0_,
```

Hibernate Query Language

```
insurance0_.invested_amount as
invested3_2_0_,
insurance0_.investement_date as
investem4_2_0_ from insurance insurance0_
where insurance0_.ID=?
First Name: Home Insurance
ID: 13
Hibernate: select insurance0_.ID as ID0_,
insurance0_.insurance_name as
insurance2_2_0_,
insurance0_.invested_amount as
invested3_2_0_,
insurance0_.investement_date as
investem4_2_0_ from insurance insurance0_
where insurance0_.ID=?
First Name: Motorcycle Insurance
ID: 14
Hibernate: select insurance0_.ID as ID0_,
insurance0_.insurance_name as
insurance2_2_0_,
insurance0_.invested_amount as
invested3_2_0_,
insurance0_.investement_date as
investem4_2_0_ from insurance insurance0_
where insurance0_.ID=?
First Name: Motorcycle Insurance
```

Here is the list of all HQL queries performed on the insurance table that you can use according to your requirement:

Clause

select

Query

```
"Select insurance.lngInsuranceId,
insurance.insuranceName," +
"insurance.investmentAmount,
insurance.investmentDate from Insurance
insurance";
```

Clause

where

Query

```
"Select insurance.lngInsuranceId,
insurance.insuranceName," + "insurance.
investmentAmount,
insurance.investmentDate from Insurance
insurance "+ " where
insurance.lngInsuranceId='1'";
```

Clause

order by

Query

```
" from Insurance as insurance order by
insurance.insuranceName";
```

Clause

group by

Query

```
"select sum(insurance.investmentAmount),
insurance.insuranceName " + "from Insurance
insurance group by
insurance.insuranceName";
```

Hibernate Update Query

```
Insurance ins =
(Insurance)sess.get(Insurance.class, new
Long(1));
ins.setInsuranceName("Travel Insurance
");
ins.setInvestementAmount(20000);
ins.setInvestementDate(new Date());
sess.update(ins);
tr.commit();
```

Hibernate Delete Query

```
String hql = "delete from Insurance insurance
where id = 2";
Query query = sess.createQuery(hql);
int row = query.executeUpdate();
```

Example list of Aggregate functions:

Funtion	Query
Avg()	"select avg(investmentAmount) from Insurance insurance"
Max()	select max (investmentAmount) fromInsurance insurance"
Min()	"select min(investmentAmount) from Insurance insurance"

Web Services - Apache Axis

Axis stands for Apache's EXTensible Interaction System. Axis is one of the most popular SOAP engine that is used to construct SOAP processor like gateways, clients, servers etc. The current version of Axis is written in java while the next client side version of Axis is being developed in C++. Axis supports to both Document-style services and RPC, therefore it seems the right right way to develop a Document-style service. Axis includes a handy tool called WSDL2Java that handles the incoming XML data that comes along with the Document-style service.

Axis is not a stand alone SOAP engine but it also contains:

- a simple stand-alone server.
- a server that plugs into servlet engines as Tomcat.
- support to the Web Service Description Language (WSDL) extensively.
- emitter tool to generate Java classes with the help of WSDL.
- some programs just to have an idea
- a tool that monitors TCP/IP packets.

While talking about the Apache's SOAP generation it is the third generation of Apache SOAP (started at IBM as "SOAP4J"). After that Apache developed an engine that was much more flexible, configurable and able to handle both SOAP as well as the upcoming XML protocol specification from the W3C. After a little time, it rearchitected and Axis now have the following key features:

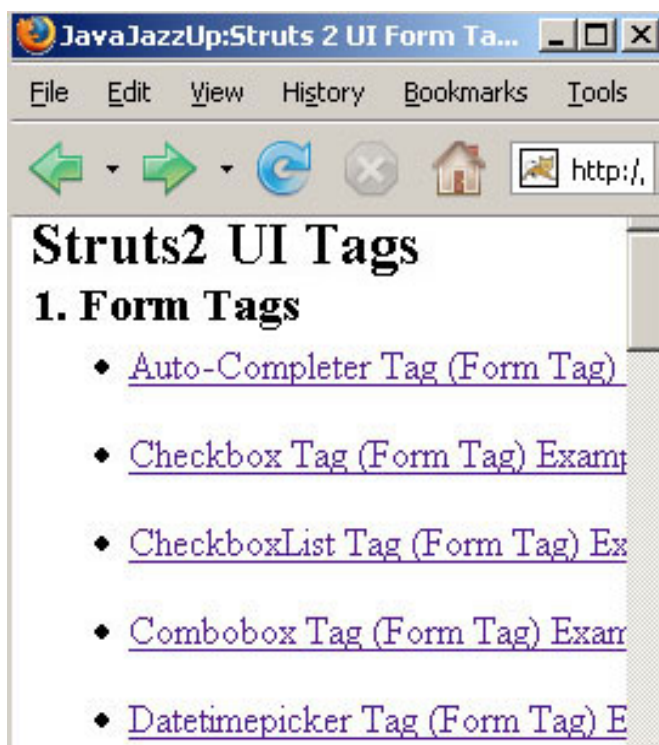
- **Transport framework:** It provides a simple abstraction to design transport (i.e. senders and listeners for SOAP over various protocols like message-oriented middleware, FTP, SMTP, etc), and core of the engine is entirely transport-independent.
- **Speed:** Axis has greater speed as compared to the earlier versions of Apache SOAP. Axis achieves this speed by using SAX (event-based) parsing.

- **Stability:** Axis introduces a set of published interfaces that enable the axis more stable than the rest of Axis as changes occurs relatively slowly.
- **Flexibility:** The Axis architecture leaves the developer completely free to insert extensions into the engine to process the system management, custom headers, or anything else you can imagine.
- **WSDL support:** Axis supports the Web Service Description Language, version 1.1. WSDL enables to the user to easily build stubs to access remote services. It also allows to the user to export machine-readable descriptions automatically of your deployed services from Axis.
- **Component-oriented deployment:** It enables the user to define reusable networks of Handlers that process the common patterns for your applications, or to distribute to partners.

Now we think that you have better understanding about Axis. It is a better way of configuring a SOAP engine.

Struts2 Tags

Apache Struts is an open-source framework used to develop Java web applications. In this section, struts 2 form tags (UITags) will be discussed and the rest will be included in the subsequent issues of the magazine. Just download the zip file "struts2UIformtags.zip" from any link given below of each page of this article, unzip it and copy this application to the webapps directory of Tomcat. Start tomcat and write `http://localhost:8080/truts2UIformtags/index.jsp` to the address bar. You can examine the result of each tag from this



1. Auto Completer Example

The autocompleter tag always displays a dropdown list with the options that have at least a partial match with entered text in the textbox. If the user clicks on the dropdown button then all options are shown in the dropdown list. The autocompleter tag generates two input fields. First is "text", whose name is specified with the "name" attribute and another one is "hidden" whose name is "\${name}. Key", where \${name} is the value in the "name"

attribute

The autocompleter tag loads its options asynchronously when the page loads suggested options based on the text entered by you in textbox. If the autoComplete attribute is set to 'true' (By default 'false') then it makes suggestions in the textbox.

Add the following code snippet into the struts.xml file.

```
<action name="autocompleter"
class="net.javajazzup.autocompleter">
    <result>/pages/formTags/
autocompleter.jsp</result>
</action>
```

Create a list in the action class and populate them with various states name of U.S. as shown in the "autocompleter" class.

```
package net.javajazzup;
import
com.opensymphony.xwork2.ActionSupport;
import java.util.*;

public class autocompleter extends
ActionSupport{
    private List state;
    public String execute() throws Exception{
        state = new ArrayList();
        state.add("Alabama");
        state.add("Alaska");
        state.add("Arizona");
        state.add("Arkansas");
        state.add("California");
        state.add("Colorado");
        state.add("Connecticut");
        state.add("Delaware");
        state.add("District of Columbia");
        state.add("Florida");
        state.add("Georgia");
        state.add("Hawaii");
        state.add("Idaho");
        state.add("Illinois");
        state.add("Indiana");
        state.add("Iowa");
        state.add("Kansas");
        state.add("Kentucky");
```

Struts2 Tags

```
state.add("Louisiana");
state.add("Maine");
state.add("Maryland");
state.add("Massachusetts");
state.add("Michigan");
state.add("Minnesota");
state.add("Mississippi");
state.add("Missouri");
state.add("Montana");
state.add("Nebraska");
state.add("Nevada");
state.add("New Hampshire");
state.add("New Jersey");
state.add("New Mexico");
state.add("New York");
state.add("North Carolina");
state.add("North Dakota");
state.add("Ohio");
state.add("Oklahoma");
state.add("Oregon");
state.add("Pennsylvania");
state.add("Rhode Island");
state.add("South Carolina");
state.add("South Dakota");
state.add("Tennessee");
state.add("Texas");
state.add("Utah");
state.add("Vermont");
state.add("Virginia");
state.add("Washington");
state.add("West Virginia");
state.add("Wisconsin");
state.add("Wyoming");
return SUCCESS;
}
public List getState(){
return state;
}
}
```

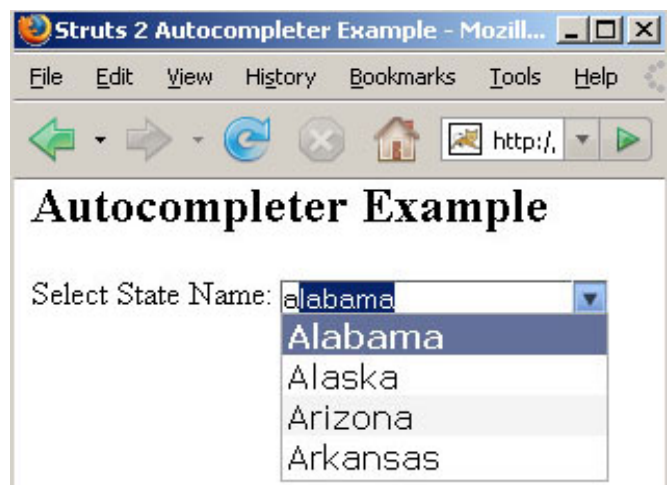
`<s:autocompleter theme="simple" list="state" StateName/>` creates a autocompleter list with the name of U.S. states.

autocompleter.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
<head>
<title>Struts 2 Autocompleter Example</
title>
```

```
<s:head theme="ajax" />
</head>
<body>
<h2>Autocompleter Example</h2>
<s:label name="stateName" value="Select
State Name:" />
<s:autocompleter theme="simple"
list="state" name="StateName"/>
</body>
</html>
```

Output of the autocompleter.jsp:



2. Checkbox Tag (Form Tag) Example

The checkbox tag is a UI tag that is used to render an HTML input element of type checkbox, populated by the specified property from the ValueStack.

Add the following code snippet into the struts.xml file.

```
<action name="checkboxTag">
<result>/pages/formTags/
checkboxTag.jsp</result>
</action>
```

Create a jsp using the tag `<s:checkbox>` It Renders an HTML input element of type checkbox.

Struts2 Tags

checkboxTag.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>

<html>
  <head>
    <title>Checkbox (Form Tag) Tag
    Example</title>
  </head>
  <body>
    <h2>Checkbox Tag Example</h2>
    <b>Sex</b><br>
    <s:checkbox label="Male" name="male"
value="true" /><br>
    <s:checkbox label="Female"
name="male" />
  </body>
</html>
```

Output of the checkboxTag.jsp:



3. Checkboxlist Tag (Form Tag) Example

The checkboxlist tag is a UI tag that creates a series of checkboxes from a list. Setup is like `<s:select />` or `<s:radio />`, but creates checkbox tags.

Add the following code snippet into the struts.xml file.

```
<action name="checkboxlistTag"
class="net.javajazzup.checkboxlistTag">
  <result>/pages/formTags/
checkboxlistTag.jsp</result>
</action>
```

Create two lists in the action class and populate them with various items as shown in the "checkboxlistTag" class.

checkboxlistTag.java

```
package net.javajazzup;
import
com.opensymphony.xwork2.ActionSupport;
import java.util.*;

public class checkboxlistTag extends
ActionSupport{

  private List fruits;
  private List animals;
  public String execute()throws Exception{
    fruits = new ArrayList();
    fruits.add("Apple");
    fruits.add("Mango");
    fruits.add("Orange");
    fruits.add("Pine Apple");

    animals = new ArrayList();
    animals.add("Dog");
    animals.add("Elephant");
    animals.add("Ox");
    animals.add("Fox");
    return SUCCESS;

  }

  public List getFruits(){
    return fruits;
  }

  public List getAnimals(){
    return animals;
  }
}
```

Create a jsp using the tag `<s:checkboxlist>`

```
<s:checkboxlist name="Fruits-name"
list="fruits" /> prints a checboxlist with name
Fruits and Creates a series of checkboxes from
fruits list of the action class "checkboxlistTag".
<s:checkboxlist name="Animals-name"
list="animals" /> prints a checboxlist with name
```

Struts2 Tags

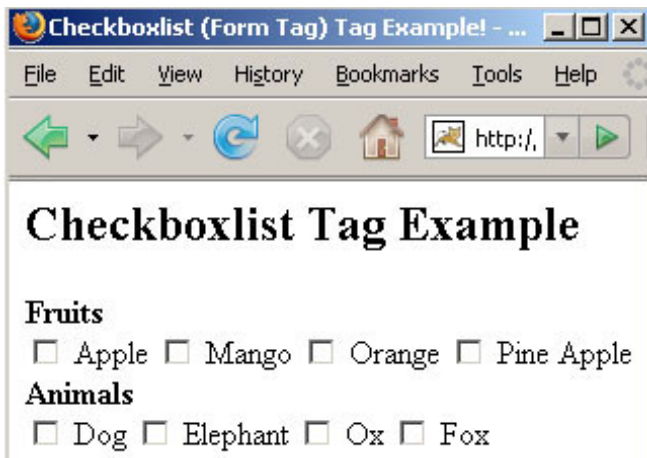
Animals and Creates a series of checkboxes from animals list of the action class "checkboxlistTag".

checkboxlistTag.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>

<html>
<head>
<title>Checkboxlist (Form Tag) Tag
Example!</title>
</head>
<body>
<h2>Checkboxlist Tag Example</h2>
<b>Fruits</b><br>
<s:checkboxlist name="Fruits-name"
list="fruits" /><br>
<b>Animals</b><br>
<s:checkboxlist name="Animals-name"
list="animals" /><br>
</body>
</html>
```

Output of the checkboxlistTag.jsp:



4. Combobox Tag (Form Tag) Example

The combo box is basically an HTML INPUT of type text and HTML SELECT grouped together to give you a combo box functionality. You can place text in the INPUT control by using the SELECT control or type it in directly in the text field.

Add the following code snippet into the struts.xml file.

```
<action name="comboboxTag"
class="net.javajazzup.comboboxTag">
<result>/pages/formTags/
comboboxTag.jsp</result>
</action>
```

Create a list in the action class and populate it with various items as shown in the "comboboxTag" class.

comboboxTag.java

```
package net.javajazzup;
import
com.opensymphony.xwork2.ActionSupport;
import java.util.*;

public class comboboxTag extends
ActionSupport{

private List fruits;
public String execute()throws Exception{
fruits = new ArrayList();
fruits.add("Apple");
fruits.add("Mango");
fruits.add("Orange");
fruits.add("Pine Apple");
return SUCCESS;

}

public List getFruits(){
return fruits;
}
}
```

Create a jsp using the tags <s:combobox>

The tag <s:combobox label="Colors Name" name="colorNames" headerValue="— Please Select —" headerKey = " 1 " list="{Black,'Green','White','Yellow','Red','Pink'}" /> prints a combobox with name color Name and an HTML INPUT of type text and HTML SELECT grouped together created using the list.

Struts2 Tags

The tag `<s:checkboxlist name="Animals-name" list="animals" />` prints a combobox with name Fruits Name and an HTML INPUT of type text and HTML SELECT grouped together created using the "fruits" list of the action class "checkboxlistTag".

comboboxTag.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>

<html>
  <head>
    <title>Combobox (Form Tag) Tag
    Example</title>
  </head>
  <body>
    <h2>Combobox Tag Example</h2>
    <s:form>
      <s:combobox label="Colors Name"
      name="colorNames"
        headerValue="— Please Select —"
        headerKey="1"
        list="{ 'Black','Green','White','Yellow',
          'Red','Pink'}" />
      <s:combobox label="Fruits Name"
      name="fruitsNames"
        headerValue="— Please Select —"
        headerKey="1" list="fruits" />
    </s:form>
  </body>
</html>
```

Output of the comboboxTag.jsp:



5. Datetimepicker Tag (Form Tag) Example

The datetimepicker tag is a UI tag that is used to render a date/time picker in a dropdown container. A stand-alone DateTimePicker widget makes it easy to select a date/time, or increment by week, month, and/or year. It is possible to customize the user-visible formatting with either the 'formatLength' (long, short, medium or full) or 'displayFormat' attributes. By default current locale will be used.

Add the following code snippet into the struts.xml file.

```
<action name="datetimepickerTag"
class="net.javajazzup.includeTag">
  <result>/pages/formTags/
datetimepickerTag.jsp</result>
</action>
```

Create an action class as shown:

includeTag.java

```
package net.javajazzup;
import
com.opensymphony.xwork2.ActionSupport;
import java.util.*;

public class includeTag extends ActionSupport
{
```

Struts2 Tags

```
private Date myBirthday;
public String execute() throws Exception{
    setMyBirthday(new Date("Jan 12, 1984
11:21:30 AM"));
    return SUCCESS;
}
public void setMyBirthday(Date date){
    this.myBirthday = date;
}
public Date getMyBirthday(){
    return myBirthday;
}
}
```

Create a jsp using the tag<s:datetimepicker>

This tag renders a date/time picker in a dropdown container.

The tag <s:datetimepicker name="myBirthday" label="My Birth Day (dd-MM-yyyy)" displayFormat="dd-MM-yyyy" /> picks the data from the action class "includeTag" using the parameter name="myBirthday" using the display format as displayFormat="dd-MM-yyyy".

datetimepickerTag.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
<head>
<title>Datetimepicker (Form Tag) Tag
Example</title>
<s:head theme="ajax" />
</head>
<body>
<h2>Datetimepicker Tag Example</h2>
<s:datetimepicker name="myBirthday"
label="Select date:(dd-MM-yyyy)"
displayFormat="dd-MM-yyyy" />
</body>
</html>
```

Output of the datetimepickerTag.jsp:

Figure6

6. Doubleselect Tag (Form Tag) Example

The doubleselect tag is a UI tag that renders two HTML select elements with second one changing displayed values depending on selected entry of first one.

Add the following code snippet into the struts.xml file

```
<action name="doubleselectTag">
<result>/pages/formTags/
doubleselectTag.jsp</result>
</action>
```

Create a jsp using the tag <s:doubleselect> This tag renders two HTML select elements with second one changing displayed values depending on selected entry of first one. This tag contains various parameters:

The headerKey parameter sets the header key of the second list. Must not be empty. In our case we have set it to "1"

The headerValue parameter sets the header value of the second list. In our case we have set it to "— Please Select —"

The doubleName parameter sets the name for complete component. In our case we have set it as : doubleName="dishes"

The doubleList sets the second iterable source to populate from. In our case we have set it as

```
:
doubleList="top == 'Color' ?
{'Black','Green','White',
'Yellow','Red','Pink'} :
'Apple','Banana','Grapes','Mango'}"
```

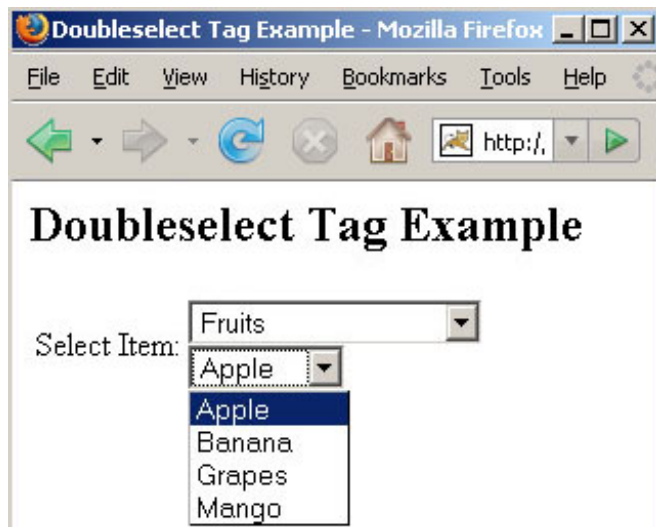
doubleselectTag.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
<head>
```

Struts2 Tags

```
<title>Doubleselect Tag Example</title>
</head>
<body>
<h2>Doubleselect Tag Example</h2>
<s:form>
<s:doubleselect label="Select Item"
  headerValue="— Please Select —"
  headerKey="1" list="{\`Color`,`Fruits`}"
  doubleName="dishes"
  doubleList="top == `Color` ?
{\`Black`,`Green`,`White`,
  `Yellow`,`Red`,`Pink`} : {
`Apple`,`Banana`,`Grapes`,`Mango`}" />
</s:form>
</body>
</html>
```

Output of the doubleselectTag.jsp:



7. File Tag (Form Tag) Example

The file tag is a UI tag that renders an HTML file input element achieved through browsing.

Add the following code snippet into the struts.xml file

```
<action name="fileTag">
  <result>/pages/formTags/fileTag.jsp</
result>
</action>
```

Create a jsp using the tag `<s:file>`. It renders an HTML file input element. The parameters name is used to set a name for element which we have used as name="uploadFile" and the parameter accept is the HTML accept attribute that indicates the accepted file mime types which we have used as accept="text/*".

fileTag.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
<head>
<title>File Tag Example</title>
</head>
<body>
<h2>File Tag Example</h2>
<b>File Name</b>
<s:form>
  <s:file name="uploadFile" accept="text/*"
/>
</s:form>
</body>
</html>
```

Output of the fileTag.jsp



File Tag Example

File Name

8. Form Tag Example

The form tag is a UI tag that renders HTML an input form. The remote form allows the form to be submitted without the page being refreshed. The results from the form can be inserted into any HTML element on the page.

Struts2 Tags

Add the following code snippet into the struts.xml file

```
<action name="formTag">
  <result>/pages/formTags/formTag.jsp</
result>
</action>
```

Create a jsp using the tag `<s:form>`. It renders HTML as an input form

formTag.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
  <head>
    <title>Form Tag Example</title>
  </head>
  <body>
    <h2>Form Tag Example</h2>
    <s:form>
      <s:textfield name="username"
label="Login name"/>
      <s:password name="password"
label="Password"/>
      <s:submit value="Login" align="center"/>
    </s:form>
  </body>
</html>
```

Output of the formTag.jsp:



Form Tag Example

Login name:

Password:

9. Label Tag (Form Tag) Example

The label tag is a UI tag that is used to render

an HTML LABEL that allow to output label:name type of combinations that has the same format treatment as the rest of UI controls.

Add the following code snippet into the struts.xml file.

```
<action name="labelTag">
  <result>/pages/formTags/labelTag.jsp</
result>
</action>
```

Create a jsp using the tag `<s:label>` It renders an HTML LABEL that allow to output `<s: label name=" " value=" " />` combination that has the same format treatment as the rest of UI controls.

labelTag.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
  <head>
    <title>Label Tag Example</title>
  </head>
  <body>
    <h2>Label Tag Example</h2>
    <s:form>
      <s:label name="name" value="Name" />
      <s:label name="roll" value="Roll" />
      <s:label name="address" value="
Address: " />
    </s:form>
  </body>
</html>
```

Struts2 Tags

Output of the labelTag.jsp:



10. Optiontransfersselect Tag (Form Tag) Example

The Optiontransfersselect tag is a UI tag that creates an option transfer select component. There are two <select ...> tags with buttons in the middle of them, which allows options in each of the <select ...> to be moved between them. It auto-selects all its elements upon its containing form submission.

Add the following code snippet into the struts.xml file.

```
<action name="optiontransfersselectTag">
  <result>/pages/formTags/
optiontransfersselectTag.jsp</result>
</action>
```

Create a jsp using the tag <s:optiontransfersselect> This tag creates an option transfer select component. This tag contains various parameters:

The **label** parameter sets label expression used for rendering a element specific label. In our case we have set it to "Employee Records"

The **name** parameter sets the name for the element. In our case we have set it to "leftSideEmployeeRecords"

The **leftTitle** parameter sets the left title. In our case we have set it to "RoseIndia"

The **rightTitle** parameter sets the right title. In

our case we have set it to "JavaJazzUp"

The **headerKey** sets the header key of the given list. It must not be empty. In our case we have set it to:"headerKey"

The **headerValue** sets the header value of the given list. In our case we have set it to:"— Please Select —"

The **doubleName** sets the name for complete component. In our case we have set it to:"rightSideEmployeeRecords"

The **doubleHeaderKey** sets the header key for the second list. In our case we have set it to:"doubleHeaderKey"

The **doubleHeaderValue** sets the header value for the second list. In our case we have set it to:"— Please Select —"

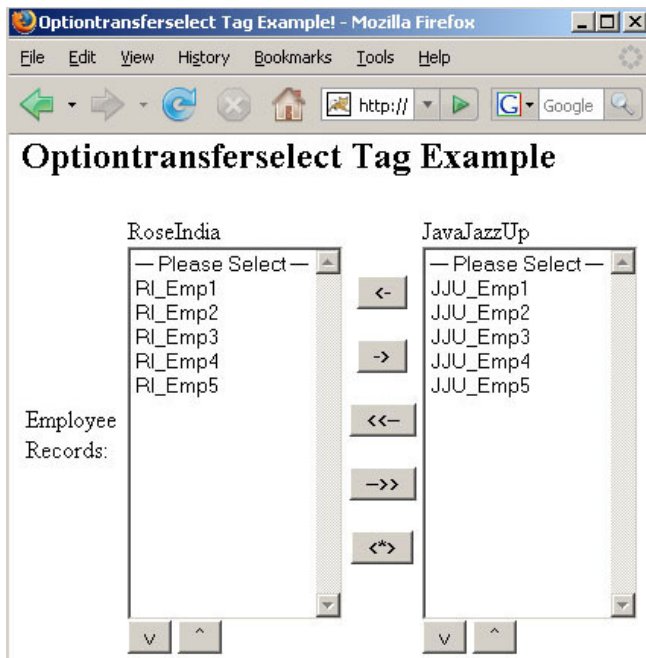
optiontransfersselectTag.jsp

```
<%@ taglib prefix="s" uri="/struts-tags"
%>
<html>
  <head>
    <title>Optiontransfersselect Tag
Example!</title>
  </head>
  <body>
    <h2>Optiontransfersselect Tag Example</
h2>
    <s:form>
      <s:optiontransfersselect
label="Employee Records"
name="leftSideEmployeeRecords"
leftTitle="RoseIndia"
rightTitle="JavaJazzUp"
list="{`RI_Emp1',
`RI_Emp2`,`RI_Emp3`,`RI_Emp4`,`RI_Emp5`}"
headerKey="headerKey"
headerValue="— Please Select —"

doubleName="rightSideEmployeeRecords"
doubleList="{`JJU_Emp1',
`JJU_Emp2`,`JJU_Emp3',
`JJU_Emp4`,`JJU_Emp5`}"
doubleHeaderKey="doubleHeaderKey"
doubleHeaderValue="— Please Select —"
/>
    </s:form>
  </body>
</html>
```

Struts2 Tags

Output of the optiontransferselctTag.jsp:



11. Optgroup Tag (Form Tag) Example

The optgroup tag is a UI tag that creates an optgroup component which needs to reside within a select tag `<s:select>`.

Add the following code snippet into the struts.xml file

```
<action name="optgroupTag">
  <result>/pages/formTags/
optgroupTag.jsp</result>
</action>
```

Create a jsp using the tag `<s:optgroup>`

Create a jsp using the tag `<s:optgroup>` within the `<s:select>` tag. It creates an optgroup component. This tag contains few parameters:

The label parameter sets the label attribute. In our case we have set it to "Hardware" and "Software".

optgroupTag.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
  <head>
    <title>Optgroup Tag Example</title>
  </head>
  <body>
    <h2>Optgroup Tag Example</h2>
    <s:form>
      <s:select label="Please Select"
name="select"

list="%{#{'PROGRAMMING': 'Programming',
'DATABASE': 'DataBase', 'WEBAPPLICATION':
'WebApplication'}}">
        <s:optgroup label="Hardware"
list="%{#{'CPU': 'Central
Processing
Unit', 'MOUSE': 'Mouse', 'KEYBOARD': 'Keyboard'}}"
/>
        <s:optgroup label="Software"
list="%{#{'SYSTEM
SOFTWARE': 'System Software', 'APPLICATION
SOFTWARE': 'Application Software'}}" />
      </s:select>
    </s:form>
  </body>
</html>
```

Struts2 Tags

Output of the optgroupTag.jsp:



12. Password Tag (Form Tag) Example

The password tag is a UI tag that renders an HTML input tag of type password.

Add the following code snippet into the struts.xml file

```
<action name="passwordTag">
  <result>/pages/formTags/
passwordTag.jsp</result>
</action>
```

Create a jsp using the tag <s:password>
It renders an HTML input tag of type password.

passwordTag.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
  <head>
    <title>Password Tag Example</title>
  </head>
  <body>
```

```
<h2>Password Tag Example</h2>
<s:form>
<s:password label="Enter Password"
name="password" size="10" maxlength="8" /
>
</s:form>
</body>
</html>
```

Output of the passwordTag.jsp:



13. Radio Tag (Form Tag) Example

The radio tag is a UI tag that renders a radio button input field.

Add the following code snippet into the struts.xml file

```
<action name="radioTag"
class="net.javajazzup.checkboxlistTag">
  <result>/pages/formTags/radioTag.jsp</
result>
</action>
```

Create an action class with two lists as shown below:

checkboxlistTag.java

```
package net.javajazzup;
import
com.opensymphony.xwork2.ActionSupport;
import java.util.*;

public class checkboxlistTag extends
```

Struts2 Tags

```
ActionSupport{  
  
    private List fruits;  
    private List animals;  
    public String execute()throws Exception{  
        fruits = new ArrayList();  
        fruits.add("Apple");  
        fruits.add("Mango");  
        fruits.add("Orange");  
        fruits.add("Pine Apple");  
  
        animals = new ArrayList();  
        animals.add("Dog");  
        animals.add("Elephant");  
        animals.add("Ox");  
        animals.add("Fox");  
        return SUCCESS;  
    }  
  
    public List getFruits(){  
        return fruits;  
    }  
  
    public List getAnimals(){  
        return animals;  
    }  
}
```

Create a jsp using the tag `<s:radio>` It renders a radio button input field.

radioTag.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>  
<html>  
    <head>  
        <title>Radio Tag Example!</title>  
    </head>  
    <body>  
        <h2>Radio Tag Example</h2>  
        <s:form>  
            <s:radio label="Fruits" name="fruitsname"  
list="fruits" />  
            <s:radio label="Animals"  
name="animalsname" list="animals" />  
        </s:form>  
    </body>  
</html>
```

Output of the radioTag.jsp:



14. Reset Tag (Form Tag) Example

The reset tag is a UI tag that is used together with the form tag to provide form resetting.

It renders a reset button.

The reset can have two different types of rendering:

input: renders as html `<input type="reset"...>`
button: renders as html `<button type="reset"...>`

The button type has advantages as it adds the possibility to separate the submitted value from the text shown on the button face.

Add the following code snippet into the struts.xml file

```
<action name="resetTag">  
    <result>/pages/formTags/resetTag.jsp</  
result>  
</action>
```

Create a jsp using the tag `<s:reset>`

It renders a reset button which provides the form resetting .

resetTag.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>  
<html>  
    <head>  
        <title>Reset Tag Example!</title>  
    </head>  
    <body>
```


Struts2 Tags

```
<h2>Reset Tag Example</h2>
<s:form>
  <s:textfield name="username" label="User
Name" size="15" maxlength="10" />
  <s:password name="password"
label="Password" size="15" maxlength="10" /
>
  <s:reset value="Reset" />
</s:form>
</body>
</html>
```

Output of the resetTag.jsp:



15. Select Tag (Form Tag) Example

The select tag is a UI tag that is used to render a HTML input tag of type select.

Add the following code snippet into the struts.xml file.

```
<action name="selectTag"
class="net.javajazzup.weekDay">
  <result>/pages/formTags/selectTag.jsp</
result>
</action>
```

Create an action class with a list populated with various items as shown below:

weekDay.java

```
package net.javajazzup;
import
com.opensymphony.xwork2.ActionSupport;
import java.util.*;

public class weekDay extends ActionSupport{

  private List day;
  public String execute()throws Exception{
    day = new ArrayList();
    day.add("Sunday");
    day.add("Monday");
    day.add("Tuesday");
    day.add("Wednesday");
    day.add("Thursday");
    day.add("Friday");
    day.add("Saturday");
    return SUCCESS;

  }

  public List getDay(){
    return day;
  }
}
```

Create a jsp using the tag <s:select> This tag creates an HTML input tag of type select. This tag contains various parameters:

The **label** parameter sets the label expression used for rendering a element specific label. In our 1st case we have set it to "Select Day" The **name** parameter sets the name for the element. In our 1st case we have set it to "daysname" The **headerKey** sets key for first item in list.. It must not be empty and wrongly specified. In both cases we have set it to:"1" The **headerValue** sets the Value expression for the first item in the list. In both cases we have set it to:"— Please Select —"

selectTag.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>

<html>
  <head>
    <title>Select Tag Example</title>
```

Struts2 Tags

```
</head>
<body>
<h2>Select Tag Example</h2>
<s:form>
  <s:select label="Select Day"
  name="daysname"
  headerKey="1"
  headerValue="-- Please Select --"
  list="day"
  />

  <s:select label="Select Month"
  name="monthname"
  headerKey="1"
  headerValue="-- Please Select --"
  list="#{'01':'January','02':'February','03':'
March','04':'April','05':'May','06':'June','07':'July',
'08':'August','09':'September','10':'
October','11':'November','12':'December'}"
  />
</s:form>
</body>
</html>
```

Output of the electTag.jsp:



16. Submit Tag (Form Tag) Example

The submit tag is a UI tag that is used to render a submit button. The submit tag is used together with the form tag to provide asynchronous form submissions. The submit can have three different types of rendering:

input: renders as html `<input type="submit"...>`
image: renders as html `<input type="image"...>`
button: renders as html `<button type="submit"...>`

Struts2 Tags

Add the following code snippet into the struts.xml file.

```
<action name="submitTag">
  <result>/pages/formTags/submitTag.jsp</
result>
</action>
```

Create a jsp using the tag `<s:submit>`
This tag renders a submit button. This tag contains various parameters:

The **value** parameter presets the value of input element. In our 1st case we have set it to "Submit".

The **type** parameter sets the types of submit to use. Valid values are input, button and image. In our case we have set it to "image"

The **src** supply an image src for image type submit button. It will have no effect for types input and button. In our case we have set it to: `"/ struts2UIformtags /pages/ formTags / submit.gif"`

The align sets HTML align attribute. In our case we have set it to: `"center"`

submitTag.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>

<html>
  <head>
    <title>Submit Tag Example</title>

  </head>
  <body>
    <h2>Submit Tag Example</h2>
    <s:form>
      <s:textfield name="username" label="User
Name" size="15" maxlength="10" />
      <s:textfield name="password"
label="Password" size="15" maxlength="10" /
>
      <s:submit value="Submit" />
      <!-- To use gif image on button -->
      <s:submit type="image" src="/
struts2UIformtags/pages/formTags/
submit.gif" align="center" />
    </s:form>
  </body>
</html>
```

Output of the submitTag.jsp:



17. Textarea Tag (Form Tag) Example

The textarea tag is a UI tag that is used to render HTML textarea.

Add the following code snippet into the struts.xml file.

```
<action name="textareaTag">
  <result>/pages/formTags/textareaTag.jsp</
result>
</action>
```

Create a jsp using the tag `<s:textarea >`
This tag renders HTML textarea tag.

`<s:textarea label="Description" name="description" cols="15" rows="10" />` tag displays a HTML textarea with label equal to Description, column value=15 and row value=10.

Struts2 Tags

textareaTag.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>

<html>
  <head>
    <title>Textarea Tag Example</title>
  </head>
  <body>
    <h2>Textarea Tag Example</h2>
    <s:form>
      <s:textarea label="Description"
name="description" cols="15" rows="10" />
    </s:form>
  </body>
</html>
```

Output of the textareaTag.jsp:



18. Textfield Tag (Form Tag) Example

The textfield tag is a UI tag that is used to render an HTML input field of type text.

Add the following code snippet into the struts.xml file.

```
<action name="textfieldTag">
  <result>/pages/formTags/textfieldTag.jsp</
result>
</action>
```

Create a jsp using the tag `<s:textfield >`
This tag renders an HTML input field of type text.

```
<s:textfield label="Employee Name"
name="empname" size="15" maxlength="10"
/> tag displays an HTML text field with label equal to Employee Name with length of 15 columns.
```

textfieldTag.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>

<html>
  <head>
    <title>Textfield Tag Example</title>
  </head>
  <body>
    <h2>Textfield Tag Example</h2>
    <s:form>
      <s:textfield label="Employee Name"
name="empname" size="15" maxlength="10"
/>
    </s:form>
  </body>
</html>
```

Output of the textfieldTag.jsp:



19. Updownselect Tag (Form Tag) Example

The updownselect tag is a UI tag that creates

Struts2 Tags

a select component with buttons to move up and down the elements in the select component . When the containing form is submitted, its elements will be submitted in the order they are arranged (top to bottom).

Add the following code snippet into the struts.xml file.

```
<action name="updownselectTag">
  <result>/pages/formTags/
updownselectTag.jsp</result>
</action>
```

Create a jsp using the tag `<s:updownselect>` This tag creates a select component with buttons to move up and down. This tag contains various parameters:

The **name** parameter presets the value of input element.. In our case we have set it to "daysname"

The **moveDownLabel** parameter is used for the text to display on the move down button. In our case we have set it to "Move Down"

The **moveUpLabel** parameter is used for the text to display on the move up button. In our case we have set it to: "Move Up"

The **selectAllLabel** parameter is used for the text to be displayed on the select all button. In our case we have set it to: "Select All"

updownselectTag.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>

<html>
  <head>
    <title>Updownselect Tag Example</title>

  </head>
  <body>
    <h2>Updownselect Tag Example</h2>
    <s:form>
      <s:updownselect

list="#{'01':'January','02':'February','03':'March','04':
'April', '05':'May','06':'June','07':'July','08':
August','09':'September',
'10':'October','11':'November',
'12':'December'}"
      name="daysname"
```

```
headerKey="1"
headerValue="— Please Select —"
moveUpLabel="Move Up"
moveDownLabel="Move Down"
selectAllLabel="Select All"
/>
</s:form>
</body>
</html>
```

Output of the updownselectTag.jsp:
Figure20

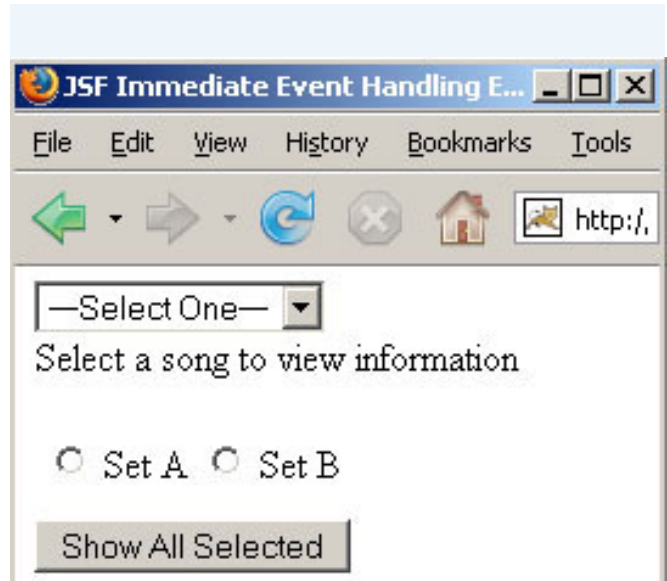
JSF Application

Immediate Event Handling Example

Event Handling is one of the important concepts in JSF. This section provides a simple JSF application, which explains how to implement "immediate" event handling in JSF. Immediate event handling is useful in cases where you do not need to validate an entire page to process a user input. Normally, the event handler for components executes in the invoke application phase. But when the "immediate" attribute is set to "true" for the component then event executes in the apply request values phase and forces JSF to skip directly to the render response phase leaving all intermediate phases of the life cycle. In this case the whole form is not validated before event handler is invoked and displays response directly.

This application takes both types of events i.e. Value Change Event and Action Event. When the user changes its choice from the list of song, a value change event occurs which displays detailed information of the selected song and when it clicks a radio button to select a category of books, a value change listener event is fired for this component which displays subcategory of the selected category i.e. displays a list of books of the selected category. When the user clicks the button labeled "Show All Selected" then an action event registered for the component is fired which displays the items selected

This application can be downloaded as zip format from the link provided in every page of the section. Extract this file and place the folder in the webapp directory of Tomcat server. Requesting the url <http://localhost:8080/jsfImmediateEventapp> through the browser will display "select.jsp" page where you can test the events. This page is shown as below.



When the user submits the above URL "index.jsp" page is called which delegates the control to the "select.jsp" page.

index.jsp

```
<html>
  <body>
    <jsp:forward page="/pages/select.jsf"/>
  </body>
</html>
```

select.jsp

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<html>
<head>
<title>JSF Immediate Event Handling Example</title>
</head>
<body>
<f:view>
<h:form id="songForm">
  <br>
  <h:selectOneMenu id="songList"
value="#{backingBean.selectedSong}"
valueChangeListener=
"#{backingBean.processValueChange1}"
immediate="true"
onchange="this.form.submit()">
```

JSF Application

```
<f:selectItems
value="#{backingBean.songList}"/>
</h:selectOneMenu>
</br>
<h:outputText id="result"
value="#{backingBean.songDetail}"/>
</br></br>
<h:selectOneRadio id="catId"
value="#{backingBean.category}"
immediate="true" onclick="submit()"
valueChangeListener=
"#{backingBean.processValueChange2}">
  <f:selectItems
value="#{backingBean.categories}"/>
  </h:selectOneRadio>
  <h:selectOneListbox id="subcategory"
value="#{backingBean.subCategory}"
binding="#{backingBean.subCategoryList}"
rendered = "false">
    <f:selectItems
value="#{backingBean.subCategories}"/>
    </h:selectOneListbox>
    </br></br>
    <h:commandButton id="cb" value="Show
All Selected" immediate="true"
actionListener=
"#{backingBean.processActionShow}">
    </br>
    <h:outputText id="showValue"
binding="#{backingBean.showAllSelected}"
rendered="true"/>
  </h:form>
</f:view>
</body>
</html>
```

In the above code, you can see that immediate attribute is set to "true" i.e. handle the events for the component in the apply request values phase rather than the invoke application phase. JavaScript code is used to process the value change event immediately when the user makes a change. Component generating action event doesn't require this support so there is no need to write JavaScript code to submit the component for processing.

The event handler methods:

The value change event handler method for the

component used for selection of the song is **processValueChange1()** which has been mentioned in the attribute **valueChangeListener**.

```
valueChangeListener=
"#{backingBean.processValueChange1}"
```

The value change event handler method for the component used for selection of category is **processValueChange2()** which has been mentioned in the attribute

valueChangeListener.

```
valueChangeListener=
"#{backingBean.processValueChange2}">
```

The action event handler method for the component used to show all selected items is **processActionShow()** which has been mentioned in the attribute **actionListener**.

```
actionListener=
"#{backingBean.processActionShow}"
```

Backing Bean:

Backing bean named "**Bean.java**" implements event handler methods. It's code is given below:

Bean.java

```
package javajazzup;

import java.util.*;
import javax.faces.component.*;
import
javax.faces.component.html.HtmlOutputText;
import
javax.faces.component.html.HtmlSelectOneListbox;
import javax.faces.event.ActionEvent;
import javax.faces.event.ValueChangeEvent;
import javax.faces.context.FacesContext;
import javax.faces.model.SelectItem;

public class Bean{
  // Stores song titles
  ArrayList songList;
```

JSF Application

```
// Stores song detail
HashMap detail;
String selectedSong;
String songDetail = "Select a song to view
information";
private String category;
private List categories;
private UIInput subCategoryList;
private String subCategory;
private List subCategories;
private HtmlOutputText showAllSelected;

public Bean() {
    songList = new ArrayList();
    //Add song titles to songList ArrayList.
    songList.add(new SelectItem("songSelect",
    "—Select One—", "s"));
    songList.add(new SelectItem("song1",
    "Song One", "s1"));
    songList.add(new SelectItem("song2",
    "Song Two", "s2"));
    songList.add(new SelectItem("song3",
    "Song Three", "s3"));

    detail = new HashMap();
    //Put song details in detail HashMap
    detail.put("songSelect", "Select a song to
    view information");
    detail.put("song1", "Album1, Singer1,
    Duration: 7.13");
    detail.put("song2", "Album2, Singer2,
    Duration: 5.20");
    detail.put("song3", "Album3, Singer3,
    Duration: 6.35");

    categories = new ArrayList();
    //Add sets to categories ArrayList.
    categories.add(new SelectItem("ch1", "Set
    A"));
    categories.add(new SelectItem("ch2", "Set
    B"));
}

public String getSelectedSong () {
    return selectedSong;
}

public void setSelectedSong (String
selectedSong) {
    this.selectedSong = selectedSong;
}

public ArrayList getSongList() {
```

```
    return songList;
}

public void setSongList(ArrayList songList)
{
    this.songList = songList;
}

public String getSongDetail() {
    return songDetail;
}

public void setSongDetail(String
songDetail){
    this.songDetail = songDetail;
}

public String getCategory(){
    return category;
}

public void setCategory(String category){
    this.category = category;
}

public String getSubCategory(){
    return subCategory;
}

public void setSubCategory(String
subCategory){
    this.subCategory = subCategory;
}

public void setCategories(List opt){
    categories = opt;
}

public List getCategories(){
    return categories;
}

public void setSubCategories(List opt){
    subCategories = opt;
}

public List getSubCategories(){
    return subCategories;
}

public void setSubCategoryList(UIInput
aSubCategoryList) {
    this.subCategoryList = aSubCategoryList;
}

public UIInput getSubCategoryList() {
    return subCategoryList;
}

public void
setShowAllSelected(HtmlOutputText
showAllSelected) {
    this.showAllSelected = showAllSelected;
}

public HtmlOutputText getShowAllSelected()
```


JSF Application

```
{
    return showAllSelected;
}

String songvalue="songSelect";
/*Value change listener method*/
public void
processValueChange1(ValueChangeEvent
vce){
    songvalue = (String) vce.getNewValue();
setSongDetail((String)detail.get(songvalue));
    /*Render the response*/

FacesContext.getCurrentInstance().renderResponse();
}

String selectchoice="No Choice";
/*Value change listener method*/
public void
processValueChange2(ValueChangeEvent
event){
    subCategoryList.setRendered(true);
    selectchoice= (String)
event.getNewValue();

    if(selectchoice.equals("ch1")){
        this.subCategories = new ArrayList();
        SelectItem suboption = new
SelectItem("subch1", "Core Java");
        subCategories.add(suboption);
        suboption = new SelectItem("subch2",
"Java Script");
        subCategories.add(suboption);
        suboption = new SelectItem("subch3",
"Ajax");
        subCategories.add(suboption);
        this.subCategories = subCategories;
    }
    if(selectchoice.equals("ch2")){
        this.subCategories = new ArrayList();
        SelectItem suboption2 = new
SelectItem("subch4", "Servlet");
        subCategories.add(suboption2);
        suboption2 = new SelectItem("subch5",
"JSP");
        subCategories.add(suboption2);
        suboption2 = new SelectItem("subch6",
"JSF");
        subCategories.add(suboption2);
    }
}
```

```
FacesContext context =
FacesContext.getCurrentInstance();
context.renderResponse();
}

/*Action listener method*/
public void
processActionShow(ActionEvent event){
    String bookvalue =
(String)subCategoryList.getSubmittedValue();
    String bookname =
getBookName(bookvalue);
    if(songvalue.equals("songSelect")){
        showAllSelected.setValue("No song
selected. "+bookname+" book selected.");
    }
    else{
        String songname =
getSongName(songvalue);
        showAllSelected.setValue(songname+"
selected. "+bookname+" book selected.");
    }
}

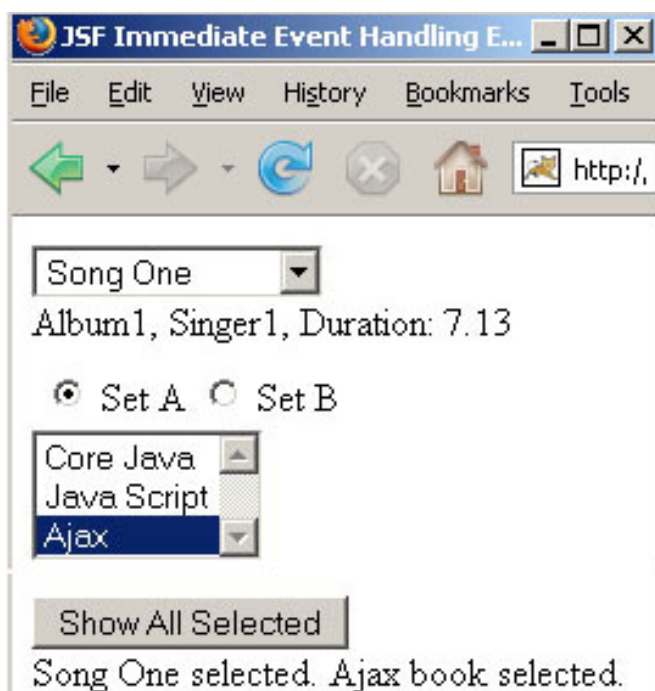
public String getBookName(String
bookvalue){
    String bookname = "No";
    if(subCategories == null){
    }
    else{
        ListIterator listItr =
subCategories.listIterator();
        while(listItr.hasNext()) {
            SelectItem si =
(SelectItem)listItr.next();
            String value = (String)(si).getValue();
            if(value.equals(bookvalue)) {
                bookname = (si).getLabel();
            }
        }
    }
    return bookname;
}

public String getSongName(String
songvalue){
    String songname="";
    for(int i=0;i<songList.size();i++){
if((((String)((SelectItem)songList.get(i)).
getValue())).equals(songvalue)){
```

JSF Application

```
songname=(String)((((SelectedItem)
(songList.get(i))).getLabel());
    }
}
return songname;
}
}
```

If the user selects "Song One" and "Ajax" from "Set A" then its output will look like:



Design Pattern

I. Observer Design Pattern

This design pattern defines one to many dependency between the objects so that if an object changes its state then all the dependent objects are notified and updated automatically. It is mainly used, to maintain consistency between objects, support to broadcast communication, to maintain classes for further use by making them loosely coupled, to make GUI application and many more. Java API provides a built-in Observable class and Observer interface.

Here we are taking an example just to demonstrate that how observer pattern works, for this, the example creates two windows. The first window takes the input from the user and the second window displays this input. As soon as the data is entered in the textfield and enter button is pressed, the second window gets the message and displays it with a dialog. The example uses the private inner class.

```
import javax.swing.*;
import java.awt.event.*;
import java.util.*;
class ShowForm extends JFrame {
    InputFormObserver inputformobserver =
    new InputFormObserver();
    InputForm inputForm ;
    Observable obsInput;
    JTextField display;
    //...
    public ShowForm() {
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
        inputForm = new InputForm();
        obsInput = inputForm.getInputInfo();
        obsInput.addObserver(inputformobserver);

        display = new JTextField(10);
        display.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
            }
        });
        getContentPane().add(display);
        setTitle("Observer form");
        setSize(200,100);
    }
}
```

```
        setLocation(200,100);
        setVisible(true);
    }

    private class InputFormObserver
    implements Observer {
        public void update(Observable ob, Object o) {
            doSomeUpdate();
            if (obsInput.countObservers()>0)
                obsInput.deleteObservers();
            obsInput = inputForm.getInputInfo();
            obsInput.addObserver(inputformobserver);
        }
    }

    public void doSomeUpdate() {
        display.setText(inputForm.getText());
        JOptionPane.showMessageDialog
        (ShowForm.this, "This form has been updated");
    }

    public static void main(String args[]) {
        ShowForm df = new ShowForm();
    }
}

class InputForm extends JFrame {
    public InputFormDisplay inform = new InputFormDisplay();
    //...
    JTextField input= new JTextField(10);
    public InputForm() {
        JPanel panel= new JPanel();
        input.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                inform.notifyObservers();
            }
        });
        panel.add(new JLabel("Enter: "));
        panel.add(input);
        addWindowListener
        (new WindowAdapter() { public void windowClosing
        (WindowEvent e) {
            System.exit(0);
        }
        });
        getContentPane().add(panel);
        setTitle("Observable form");
        setSize(200,100);
        setVisible(true);
    }
    public Observable getInputInfo() {
        return inform;
    }
}
```

Design Pattern

```
public String getText() {
    return input.getText();
}

private class InformDisplay extends Observable {
    public void notifyObservers() {
        setChanged();
        super.notifyObservers();
    }
    public String getChange() {
        return input.getText();
    }
}
//...}
```

As we pass the command "java Show" on the command prompt, two windows are displayed. In which the first window takes the input from the user and the other window displays the inputted data.

II.State Design Pattern

The State pattern is used whenever an enclosing class switches among the number of related contained classes and passes the method calls on the current contained class. This design pattern switches between internal classes in such a manner that the enclosing object appears to change its state. This design pattern also provides memory for the instance variables of a class. It localizes the state-specific behavior and partitions behavior for different states. It makes the explicit transition.

Here is an example that demonstrate State Design Pattern.

StateContext.java

```
public class StateContext {
    private StateName StateName;

    public StateContext() {
        setStateName(new StateNameStars());
        //start with stars
    }

    public void setStateName(StateName StateNameIn) {
        this.StateName = StateNameIn;
    }
}
```

```
public void showName(String nameIn) {
    this.StateName.showName(this, nameIn);
}
}
```

StateName.java

```
public interface StateName {
    public void showName(StateContext StateContext,
        String nameIn);
}
```

StateNameExclaim.java

```
public class StateNameExclaim
implements StateName {
    public StateNameExclaim() {}

    public void showName(StateContext StateContext,
        String nameIn) {
        System.out.println(nameIn.replace(' ','!'));
        //
        show exclaim only once, switch back to stars
        StateContext.setStateName(new StateNameStars());
    }
}
```

StateNameStars.java

```
public class StateNameStars implements
StateName {
    int starCount;

    public StateNameStars() {
        starCount = 0;
    }

    public void showName(StateContext StateContext,
        String nameIn) {
        System.out.println(nameIn.replace(' ','*'));
        //
        show stars twice, switch to exclamation point
        if (++starCount > 1) {
            StateContext.setStateName(
                new StateNameExclaim());
        }
    }
}
```

Design Pattern

TestState.java

```
class TestState {
    public static void main(String[] args) {
        StateContext stateContext = new StateContext();
        stateContext.showName(
            "Mr. Deepak Kumar - "+
            "IT Manager Roseindia Technologies");
        stateContext.showName(
            "Mr. Amit Kumar: Operational Manager
Roseindia Technologies");
        stateContext.showName(
            "Mr. Aquil Ahmad Khan: HR
Manager: Roseindia Technologies");
        stateContext.showName(
            "And this is me, Mohd. Zulfiqar Ahmed:
Senior Software Engineer
Roseindia Technologies");
    }
}
```

Here is the Output:

```
C:\DP\State>java
TestState      Mr.*Deepak*Kumar*-
*IT*Manager*Roseindia*Technologies
Mr.*Amit*Kumar:*Operational*Manager
*Roseindia*Technologies
And*this*is*me,*Mohd.*Zulfiqar*Ahmed:*Senior
*Software*Engineer*Roseindia*Technologies
```

III.Strategy pattern

The Strategy pattern is a design pattern in which an object controls which of a family of methods is called simply. Each method of this family stays in its own class and extends a common base class. This method introduces a family of algorithms, encapsulates them with each other and makes them interchangeable. This pattern provides pluggable behavior that enforces the client access to services. This pattern is used in those situations where many related classes differs only in their behavior. Here we are taking an example that encapsulates several algorithms (classes) into a single module just to provides the alternatives. These alternatives are provided by generating random numbers to serve the purpose.

```
interface StrategyPattern {
```

```
    public void print();
}
class GetIt implements StrategyPattern {
    public void print() {
        System.out.println("Still you are not late!");
    }
}
class StartNow implements StrategyPattern {
    public void print() {
        System.out.println("You can also start from now!");
    }
}
class LooseNothing implements StrategyPattern {
    public void print() {
        System.out.println
("You didn't loose anything so
get up and start now!");
    }
}
class Dice {
    public int throwIt() {
        return (int)(Math.random()*6)+1;
    }
}
class Test {
    static void goodLuck() {
        int yourluckyNum = new Dice().throwIt();
        StrategyPattern sp;
        switch (yourluckyNum) {
            case 2: sp = new GetIt();
                break;
            case 5: sp = new StartNow();
                break;
            default: sp = new LooseNothing();
        }
        sp.print();
    }
    public static void main(String[] args) {
        goodLuck();
    }
}
```

Here is the output:

```
C:\DP1\Strategy Pattern>java Test
You didn't loose anything so get up and start
now!
C:\DP1\Strategy Pattern>java Test
Still you are not late!
```

Design Pattern

```
C:\DP1\Strategy Pattern>java Test  
You can also start from now!
```

```
C:\DP1\Strategy Pattern>java Test  
You didn't loose anything so get up and start  
now!
```

Tips & Tricks

1. Send data from database in PDF file as servlet response:

This example retrieves data from MySQL and sends response to the web browser in the form of a PDF document using Servlet. This program uses iText, which is a java library containing classes to generate documents in PDF, XML, HTML, and RTF. For this you need to place iText.jar file to lib folder of your JDK and set classpath for it. You can download this jar file from the link [http:// www.lowagie.com/iText/download.html](http://www.lowagie.com/iText/download.html) The program below will embed data retrieved from database in PDF document.

ShowPdf.java

```
import java.io.*;
import java.util.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;
import com.lowagie.text.*;
import com.lowagie.text.pdf.*;

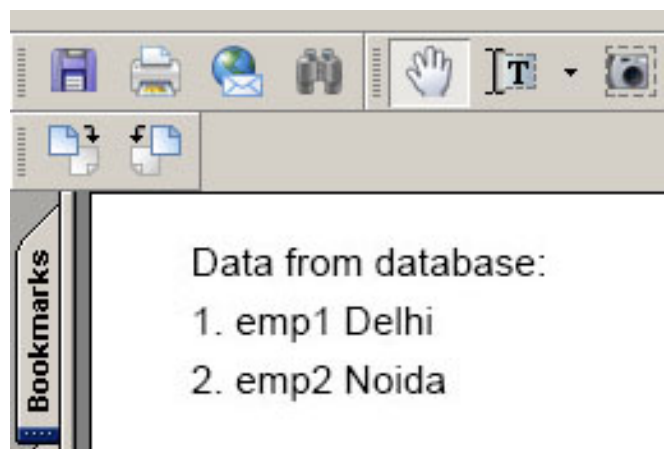
public class ShowPdf extends HttpServlet{
    public void doGet(HttpServletRequest req,
        HttpServletResponse res) throws
        ServletException, IOException{
        res.setContentType("application/pdf");
        //Create a document-object
        Document document = new Document();
        try{
            Class.forName("com.mysql.jdbc.Driver");
            Connection con =
                DriverManager.getConnection("jdbc:mysql://
                localhost:3306/test", "root", "root");

            java.util.List list = new ArrayList();
            Statement st = con.createStatement();
            ResultSet rs = st.executeQuery("select *
            from employee");
            while(rs.next()){
                String id = rs.getString("empId");
                String add = rs.getString("empAdd");
                list.add(id+" "+add);
            }
            // Create a PDF document writer
            PdfWriter.getInstance(document,
            res.getOutputStream());
            document.open();
```

```
// Create paragraph
Paragraph paragraph = new
Paragraph("Data from database:");
// Add paragraph to the document
document.add(paragraph);

com.lowagie.text.List list1=new
com.lowagie.text.List(true);
Iterator i = list.iterator();
while(i.hasNext()){
    list1.add(new
    ListItem((String)i.next()));
}
// Add list of data retrieved from
database to the document
document.add(list1);
document.close();
}
catch (Exception e) {
    e.printStackTrace();
    // Close the document
    document.close();
}
}
```

Output of the program:



2. Viewing html source of a page running on the server:

You can view complete html code of a page running on the server with the help of Java. This section provides you the complete code of the above program. This program takes a url

Tips & Tricks

starting from "http://" otherwise it sends an error message indicating invalid url.

ViewSource.java

```
import java.io.*;
import java.net.*;

public class ViewSource {
    public static void main (String[] args) throws
    IOException{
        System.out.print("Enter url to view html
source code:");
        BufferedReader br = new
BufferedReader(new
InputStreamReader(System.in));
        String url = br.readLine();
        try{
            URL u = new URL(url);
            HttpURLConnection uc =
(HttpURLConnection) u.openConnection();
            int code = uc.getResponseCode();
            String response =
uc.getResponseMessage();
            System.out.println("HTTP/1.x " + code +
" " + response);
            for(int j = 1; ; j++){
                String header = uc.getHeaderField(j);
                String key = uc.getHeaderFieldKey(j);
                if(header == null || key == null)
                    break;
                System.out.println(uc.getHeaderFieldKey(j)
+ ": " + header);
            }
            InputStream in = new
BufferedInputStream(uc.getInputStream());
            Reader r = new InputStreamReader(in);
            int c;
            while((c = r.read()) != -1){
                System.out.print((char)c);
            }
        }
        catch(MalformedURLException ex){
            System.err.println(url + " is not a valid
URL.");
        }
        catch(IOException ie){
            System.out.println("Input/Output Error: "
+ ie.getMessage());
        }
    }
}
```

In the program, HttpURLConnection is the abstract class of the java.net package. This class extends the URLConnection class of the java.net package and facilitates all same facilities of the URLConnection class with the specific HTTP features specially. getResponseCode() method of the HttpURLConnection class returns an integer value which is the code for the status from the http response message. getResponseMethod() method of the HttpURLConnection class shows the message with the http response code from the server. getHeaderFieldKey(int j) method returns the key for the given jth header field.

Output of the program:

```
C:\JavaJazzUp>java ViewSource
Enter url of local for viewing html source
code: http://localhost:8080/JJU/index
.jsp
HTTP/1.x 200 OK
Server: Apache-Coyote/1.1
Set-Cookie: JSESSIONID=
CC15E9CA0342CB43B2E2A3352E657528;
Path=/JJU
Content-Type: text/html;charset=ISO-8859-
1
Content-Length: 205
Date: Tue, 08 Jan 2008 05:59:07 GMT
<!DOCTYPE html PUBLIC "-//W3C//DTD
XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd"
>
<html>
<head>
    <title>Index</title>
</head>
<body>
Hello.....
</body>
</html>
```

3. Showing digital clock in Applet

This is an example of applet showing current time as in digital clock. This type of program is used to display the time on browser where your application is running.

In this example the time will be displayed in an

Tips & Tricks

applet in the time format like: hours: minutes: seconds AM/PM (hh:mm:ss AM/PM). Here, the ClockApplet class name extends from the Applet class and implements to the Runnable interface. start() method creates a new instance of the Thread class and starts it. Loop in run() method lets the applet repaint itself till the current thread is not null and tells the thread to sleep for 1 second. repaint() method calls the applet's paint() method which updates the applet.

ClockApplet.java

```
import java.applet.*;
import java.awt.*;
import java.util.*;

public class ClockApplet extends Applet
implements Runnable{
    Thread t,t1;
    public void start(){
        t = new Thread(this);
        t.start();
    }

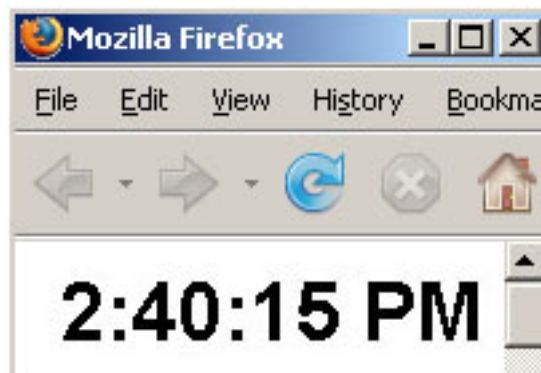
    public void run(){
        t1 = Thread.currentThread();
        while(t1 == t){
            repaint();
            try{
                t1.sleep(1000);
            }catch(InterruptedException e){}
        }
    }

    public void paint(Graphics g){
        Calendar cal = new GregorianCalendar();
        String hour =
String.valueOf(cal.get(Calendar.HOUR));
        String minute =
String.valueOf(cal.get(Calendar.MINUTE));
        String second =
String.valueOf(cal.get(Calendar.SECOND));
        int int_am_pm = cal.get(Calendar.AM_PM);
        String str_am_pm = "AM";
        if(int_am_pm==1)
            str_am_pm = "PM";

        Font f = new Font("SansSerif", Font.BOLD,
50);
        g.setFont(f);
```

```
        g.drawString(hour + ":" + minute + ":" +
second + " " + str_am_pm, 50, 50);
    }
}
```

Output of the program:



4. Get Column Names From ResultSet in MySQL

In JDBC, ResultSet is used to retrieve column values using either the index number or the name of the column. ResultSet is generated by executing a statement that queries the database. If we want to find out name, type, properties of the columns retrieved through ResultSet object then ResultSetMetaData object is used which is returned by the getMetaData() of ResultSet. In the example below, ResultSet contains values of all the columns of the "employee" table of "test" database. The following code fragment creates the ResultSet object rs, creates the ResultSetMetaData object rsmd, and uses rsmd to find out number of columns rs has, types of columns and lengths of columns, table name of the column. List of tables in test database used in the example has been given below:

Tables in test database in MySQL:

employee table:

empId	empAdd
emp1	Delhi
emp2	Noida

Tips & Tricks

workexp table:

empId	technology years
emp1	Java 4
emp1	php 2

The program, given below, uses jdbc code to store the data and metadata for the columns of the tables in "ResultSet" and "ResultSetMetaData" objects.

GetMetadata.java

```
import java.sql.*;

public class GetMetadata {
    public static void main(String[] args) throws
    Exception {
        Class.forName("com.mysql.jdbc.Driver");
        Connection con =
        DriverManager.getConnection("jdbc:mysql://
        localhost:3306/test", "root", "root");
        Statement st = con.createStatement();
        ResultSet rs = st.executeQuery("select
        employee.empId, workexp.technology from
        employee, workexp where employee.empId =
        workexp.empId");
        getColumnNames(rs);
        rs.close();
        st.close();
        con.close();
    }

    public static void
    getColumnNames(ResultSet rs) throws
    SQLException {
        if (rs == null) {
            return;
        }
        ResultSetMetaData rsmd =
        rs.getMetaData();
        // Get the number of columns
        int numberOfColumns =
        rsmd.getColumnCount();

        for (int i = 1; i < numberOfColumns + 1;
        i++) {
            System.out.println();
            // get the column name at ith index
            System.out.println("Column Name
            :"+rsmd.getColumnName(i));
        }
    }
}
```

```
// get the Data Type of the column
System.out.println("Data Type
:"+rsmd.getColumnTypeName(i));
// get the Length of the column
System.out.println("Length
:"+rsmd.getColumnDisplaySize(i));
// Get the table name of the column
System.out.println("Table Name
:"+rsmd.getTableName(i));
    }
}
}
```

Output of the program:

```
C:\JavaJazzUp>java GetMetadata
Column Name :empId
Data Type :VARCHAR
Length :6
Table Name :employee

Column Name :technology
Data Type :VARCHAR
Length :20
Table Name :workexp

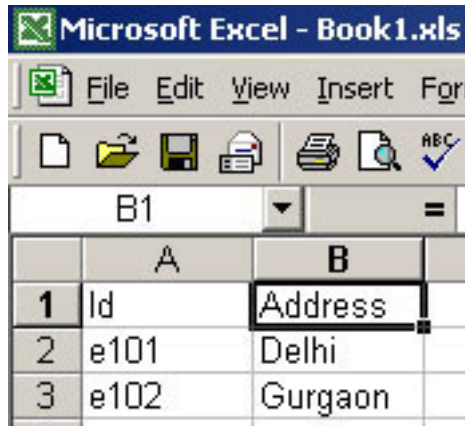
C:\JavaJazzUp>
```

5. Copy data from Excel to the MySQL

You may want to copy data in excel sheet to the table of MySQL database. This JDBC program can help you to understand how it can be done in java. ExcelToMySQL. Before running this program you have to make dsn. Open the odbc data source administrator console, create new data source, select microsoft excel driver, give data source name (dsn) and select excel sheet name. In this example, the dsn name is "myexcel". Two connections have been created, one for excel and the other for MySQL. Three queries are fired, first one to get the data from excel sheet, the next to create the table of your choice, if doesn't exist, with all the columns and name as in excel sheet and the last one to insert the data to the specified table.

Tips & Tricks

The data in the excel sheet can be seen below:



	A	B
1	Id	Address
2	e101	Delhi
3	e102	Gurgaon

This is the program below, which will copy the above data to the MySQL.

ExcelToMySQL.java

```
import java.sql.*;
import java.util.*;
import java.io.*;

public class ExcelToMySQL {
    public static void main(String args[]) throws
    IOException{
        System.out.println("Enter table name:");
        BufferedReader bf = new
        BufferedReader(new
        InputStreamReader(System.in));
        String tableName = bf.readLine();

        Connection con_excel = null, con_mysql =
        null;
        Statement stmt_excel = null, stmt_mysql
        = null;
        ResultSet rs_excel = null;
        List columnNameList = null;
        try {
            con_excel =
            getConnection("sun.jdbc.odbc.JdbcOdbcDriver",
            "jdbc:odbc:myexcel", "", "");
            stmt_excel =
            con_excel.createStatement();
            String query_excel = "select * from
            [Sheet1$]";
            rs_excel =
            stmt_excel.executeQuery(query_excel);
            columnNameList =
            getColumnNameList(rs_excel);
```

```
            con_mysql =
            getConnection("com.mysql.jdbc.Driver",
            "jdbc:mysql://localhost:3306/test", "root",
            "root");
            stmt_mysql =
            con_mysql.createStatement();
            String query_mysql =
            getQueryStringToCreateTable(tableName,
            columnNameList);

            stmt_mysql.executeUpdate(query_mysql);
            PreparedStatement p_stmt_mysql =
            con_mysql.prepareStatement
            (getQueryStringToInsertValues(tableName,
            columnNameList));
            insertValuesAndExecuteQuery
            (rs_excel, columnNameList, p_stmt_mysql);
        }
        catch (Exception e) {
            System.err.println(e.getMessage());
        }
        finally {
            try {
                rs_excel.close();
                stmt_excel.close();
                stmt_mysql.close();
                con_excel.close();
                con_mysql.close();
            }
            catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }

    public static Connection
    getConnection(String driver, String url, String
    username, String password) throws
    Exception {
        Class.forName(driver);
        return DriverManager.getConnection(url,
        username, password);
    }

    public static List
    getColumnNameList(ResultSet rs) throws
    SQLException{
        List list = new ArrayList();
        ResultSetMetaData rsmd =
        rs.getMetaData();
        int numberOfColumns =
        rsmd.getColumnCount();
        for (int i = 1; i < numberOfColumns + 1;
        i++) {
```

Tips & Tricks

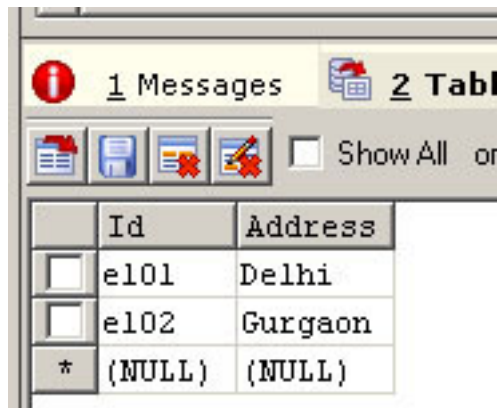
```
list.add(rsmd.getColumnName(i));
}
return list;
}

public static String
getQueryStringToCreateTable(String
tableName, List columnNameList){
    Iterator it = columnNameList.iterator();
    String st = "create table "+ tableName+"("";
    for(int i=0; i<columnNameList.size();i++){
        if(i!=0){
            st = st+",";
        }
        st= st+(String)columnNameList.get(i);
        st = st+" varchar(10);
    }
    st = st+"))";
    return st;
}

public static String
getQueryStringToInsertValues(String
tableName, List columnNameList){
    Iterator it = columnNameList.iterator();
    String st = "insert into "+ tableName+"("";
    String values = "values("";
    for(int i=0; i<columnNameList.size();i++){
        if(i!=0){
            st = st+",";
            values = values+",";
        }
        st= st+(String)columnNameList.get(i);
        values = values +"?";
    }
    st = st+"))"+ values+"))";
    return st;
}

public static void
insertValuesAndExecuteQuery(ResultSet rs,
List columnNameList, PreparedStatement
p_stmt_mysql) throws SQLException{
    while (rs.next()) {
        for(int i=0;
i<columnNameList.size();i++){
            p_stmt_mysql.setString(i+1,
rs.getString((String)columnNameList.get(i)));
        }
        int n = p_stmt_mysql.executeUpdate();
    }
}
}
```

The data in MySQL can be seen below:



The screenshot shows a MySQL database interface with a table named 'Table'. The table has two columns: 'Id' and 'Address'. The data rows are as follows:

	Id	Address
<input type="checkbox"/>	e101	Delhi
<input type="checkbox"/>	e102	Gurgaon
*	(NULL)	(NULL)

Advertise with JavaJazzUp

We are the top most providers of technology stuffs to the java community. Our technology portal network is providing standard tutorials, articles, news and reviews on the Java technologies to the industrial technocrats. Our network is getting around 3 million hits per month and its increasing with a great pace.

For a long time we have endeavored to provide quality information to our readers. Furthermore, we have succeeded in the dissemination of the information on technical and scientific facets of IT community providing an added value and returns to the readers. We have serious folks that depend on our site for real solutions to development problems.

JavaJazzUp Network comprises of :

- <http://www.roseindia.net>
- <http://www.newstrackindia.com>
- <http://www.javajazzup.com>
- <http://www.allcooljobs.com>

Advertisement Options:

Banner	Size	Page Views	Monthly
Top Banner	470*80	5,00,000	USD 2,000
Box Banner	125 * 125	5,00,000	USD 800
Banner	460x60	5,00,000	USD 1,200
Pay Links		Un Limited	USD 1,000
Pop Up Banners		Un Limited	USD 4,000

The <http://www.roseindia.net> network is the "real deal" for technical Java professionals. Contact me today to discuss your customized sponsorship program. You may also ask about advertising on other Technology Network.

Deepak Kumar
deepak@roseindia.net

India's Cheapest web Service Provider

Web Packages

Package (in INR)

Package Name	Price
Starter Package	Rs 5,100 + Taxes Extra
Business Package	Rs 9,111 + Taxes Extra
Corporate Package	Rs 22,500 + Taxes Extra
Smart Package	Rs 45,500 + Taxes Extra

* Domain Registration free with every package

Packages Specifications

Starter Package

- 5 Web Pages
- 10 Stock Images
- 5 POP 3 Accounts
- 50 MB Hosting Space
- Unlimited Edits

Business Package

- 10 Web Page Design
- Flash Site Animation
- 25 Stock Images
- 10 POP3 Accounts
- 50 MB Hosting Space
- Unlimited Edits

Corporate Package

- 25 Web Page Design
- Flash Site Animation
- Flash Intro Page
- 50 Stock Images
- 25 POP3 Accounts
- 100 MB Hosting Space
- Unlimited Edit

Smart Package

- 200 Web Page Design
- Catalog with 200 items
- Flash Site Animation
- Flash Intro Page
- Unlimited Stock Images
- 50 POP3 Accounts
- 250 MB Hosting Space
- Unlimited Edits



Logon to: <http://www.roseindia.net/services/>

Valued JavaJazzup Readers Community

We invite you to post Java-technology oriented stuff. It would be our pleasure to give space to your posts in JavaJazzup.

Contribute to Readers Forum

If theres something youre curious about, were confident that your curiosity, combined with the knowledge of other participants, will be enough to generate a useful and exciting Readers Forum. If theres a topic you feel needs to be discussed at JavaJazzup, its up to you to get it discussed.

Convene a discussion on a specific subject

If you have a topic youd like to talk about . Whether its something you think lots of people will be interested in, or a narrow topic only a few people may care about, your article will attract people interested in talking about it at the Readers Forum. If you like, you can prepare a really a good article to explain what youre interested to tell java technocrats about.

Sharing Expertise on Java Technologies

If youre a great expert on a subject in java, the years you spent developing that expertise and want to share it with others. If theres something youre an expert on that you think other technocrats might like to know about, wed love to set you up in the Readers Forum and let people ask you questions.

Show your innovation

We invite people to demonstrate innovative ideas and projects. These can be online or technology-related innovations that would bring you a great appreciations and recognition among the java technocrats around the globe.

Hands-on technology demonstrations

Some people are Internet experts. Some are barely familiar with the web. If you'd like to show others aroud some familiar sites and tools, that would be great. It would be our pleasure to give you a chance to provide your demonstrations on such issues : How to set

up a blog, how to get your images onto Flickr, How to get your videos onto YouTube, demonstrations of P2P software, a tour of MySpace, a tour of Second Life (or let us know if there are other tools or technologies you think people should know about...).

Present a question, problem, or puzzle

Were inviting people from lots of different worlds. We do not expect everybody at Readers Forum to be an expert in some areas. Your expertise is a real resource you may contribute to the Java Jazzup. We want your curiosity to be a resource, too. You can also present a question, problem, or puzzle that revolves around java technologies along with their solution that you think would get really appreciated by the java readers around the globe.

Post resourceful URLs

If you think you know such URL links which can really help the readers to explore their java skills. Even you can post general URLs that you think would be really appreciated by the readers community.

Anything else

If you have another idea for something youd like to do, talk to us. If you want to do something that we havent thought of, have a crazy idea, wed really love to hear about it. Were open to all sorts of suggestions, especially if they promote readers participation.

JOY OF WORK



All Cool Jobs
<http://www.allcooljobs.com/>

Original Perception



CBI's failure to act against influential

For some time judiciary has pronounced such welcoming judgements which have been instrumental in enhancing the faith of people in the criminal judicial system of the country. But there is one agency which has often failed to deliver its honest services on this front. The central investigating agency CBI has been chided by the court in many cases for not dealing with the case seriously especially when the accused ones are from the influential background mainly from political and administrative.