Index
Struts 2 Features

The strut-2 framework is designed for the compilation of the entire development cycle including building, developing and maintaining the whole application. It is very extensible as each class of the framework is based on an Interface and all the base classes are given an extra application and even you can add your own. The basic platform requirements are Servlet API 2.4, JSP API 2.0 and Java 5.

Some of the general features of the current Apache Strut 2 framework are given below.

Architecture – First the web browser request a resource for which the Filter Dispatcher decides the suitable action. Then the Interceptors use the required functions and after that the Action method executes all the functions like storing and retrieving data from a database. Then the result can be seen on the output of the browser in HTML, PDF, images or any other.

Tags - Tags in Strut 2 allow creating dynamic web applications with less number of coding. Not only these tags contain output data but also provide style sheet driven markup that in turn helps in creating pages with less code. Here the tags also support validation and localization of coding that in turn offer more utilization. The less number of codes also makes it easy to read and maintain.

MVC – The Model View Controller in Strut 2 framework acts as a coordinator between application’s model and web view. Its Controller and View components can come together with other technology to develop the model. The framework has its library and markup tags to present the data dynamically.

Configuration – Provides a deployment descriptor to initialize resources in XML format. The initialization takes place simply by scanning all the classes using Java packages or you can use an application configuration file to control the entire configuration. Its general-purpose defaults allow using struts directly Out of the box.

Configuration files are re-loadable that allows changes without restarting a web container.

Other Features:

- All framework classes are based on interfaces and core interfaces are independent from HTTP.
- Check boxes do not require any kind of special application for false values.
- Any class can be used as an action class and one can input properties by using any JavaBean directly to the action class.
- Strut 2 actions are Spring friendly and so easy to Spring integration.
- AJAX theme enables to make the application more dynamic.
• Portal and servlet deployment are easy due to automatic portlet support without altering code.
• The request handling in every action makes it easy to customize, when required.

**Struts 2 History**

Apache Struts is an open-source framework that is used for developing Java web application. Originally developed by the programmer and author Craig R. McClanahan, this was later taken over by the Apache Software Foundation in 2002. Struts have provided an excellent framework for developing application easily by organizing JSP and Servlet based on HTML formats and Java code. Strut1 with all standard Java technologies and packages of Jakarta assists to create an extensible development environment. However, with the growing demand of web application, Strut 1 does not stand firm and needs to be changed with demand. This leads to the creation of Strut2, which is more developer friendly with features like Ajax, rapid development and extensibility.

Struts is a well-organized framework based on **MVC** architecture. In **Model-View-Controller Architecture**, Model stands for the business or database code, View represents the page design code and the Controller for navigational code. All these together makes Struts an essential framework for building Java applications. But with the development of new and lightweight MVC based frameworks like Spring, Stripes and Tapestry, it becomes necessary to modify the Struts framework. So, the team of Apache Struts and another J2EE framework, **WebWork of OpenSymphony** joined hand together to develop an advanced framework with all possible developing features that will make it developer and user friendly.

**Strut2** contains the combined features of Struts Ti and WebWork 2 projects that advocates higher level application by using the architecture of WebWork2 with features including a plugin framework, a new API, Ajax tags etc. So the Struts communities and the WebWork team brought together several special features in WebWork2 to make it more advance in the Open Source world. Later the name of WebWork2 has changed to Struts2. Hence, Apache Strut 2 is a dynamic, extensible framework for a complete application development from building, deploying and maintaining.

WebWork is a framework for web-application development that has been included in Struts framework 2.0 release. It has some unique concepts and constructs like its compatibility of working within existing Web APIs in Java rather than trying to replace them completely. It has been built specifically taking into account the developer’s productivity and code simplicity. Furthermore it is completely context dependent that provides a wrapper around XWork. When working on web applications the web work provides a context that helps web developer in specific implementations.

While, XWork provides a mechanism that is used for configuration and factory implementation management. This mechanism is dependencies inject mechanism.
Struts 2 Architecture

Struts and webwork has joined together to develop the Struts 2 Framework. Struts 2 Framework is very extensible and elegant for the development of enterprise web application of any size. In this section we are going to explain you the architecture of Struts 2 Framework.

Request Lifecycle in Struts 2 applications

1. **User Sends request:** User sends a request to the server for some resource.

2. **FilterDispatcher determines the appropriate action:** The FilterDispatcher looks at the request and then determines the appropriate Action.

3. **Interceptors are applied:** Interceptors configured for applying the common functionalities such as workflow, validation, file upload etc. are automatically applied to the request.

4. **Execution of Action:** Then the action method is executed to perform the database related operations like storing or retrieving data from the database.

5. **Output rendering:** Then the Result renders the output.

6. **Return of Request:** Then the request returns through the interceptors in the reverse order. The returning request allows us to perform the clean-up or additional processing.

7. **Display the result to user:** Finally the control is returned to the servlet container, which sends the output to the user browser.
Struts 2 high level overview of request processing:

**Struts 2 Architecture**

Struts 2 is a very elegant and flexible front controller framework based on many standard technologies like Java Filters, Java Beans, ResourceBundles, XML etc.

For the **Model**, the framework can use any data access technologies like JDBC, EJB, Hibernate etc and for the **View**, the framework can be integrated with JSP, JTL, JSF, Jakarta Velocity Engine, Templates, PDF, XSLT etc.

**Exception Handling:**

The Struts 2 Framework allows us to define exception handlers and inceptors.

- **Exception Handlers:**
  Exception handlers allows us to define the exception handling procedure on global and local basis. Framework catches the exception and then displays the page of our choice with appropriate message and exception details.

- **Interceptors:**
  The Interceptors are used to specify the "request-processing lifecycle" for an action. Interceptors are configured to apply the common functionalities like workflow, validation etc.. to the request.

**Struts 2 Architecture**
The following diagram depicts the architecture of Struts 2 Framework and also shows the initial request goes to the servlet container such as tomcat, which is then passed through standard filter chain.

**Image: Struts 2 Architecture**

The filter chain includes:

- **Action ContextCleanUp filter:**
  The ActionContextCleanUp filter is optional and it is useful when integration has to be done with other technologies like SiteMash Plugin.

- **FilterDispatcher:**
  Next the FilterDispatch is called, which in turn uses the ActionMapper to determine whether to invoke an Action or not. If the action is required to be invoked, the FilterDispatcher delegates the control to the **ActionProxy**.

- **ActionProxy:**
  The ActionProxy takes help from Configuration Files manager, which is initialized from the struts.xml. Then the ActionProxy creates an
**ActionInvocation**, which implements the command pattern. The
ActionInvocation process invokes the Interceptors (if configured) and then
invokes the action. The the ActionInvocation looks for proper result. Then the
result is executed, which involves the rendering of JSP or templates.

Then the Interceptors are executed again in reverse order. Finally the response
returns through the filters configured in web.xml file. If the
ActionContextCleanUp filter is configured, the FilterDispatcher does not clean
the ThreadLocal ActionContext. If the ActionContextCleanUp filter is not present
then the FilterDispatcher will cleanup all the ThreadLocals present.

In this section we have learnt about the Architecture of Struts 2 Framework.

**Why Struts 2**

The new version Struts 2.0 is a combination of the Sturts action framework and
Webwork. According to the Struts 2.0.1 release announcement, some key features
are:

- **Simplified Design** - Programming the abstract classes instead of interfaces is one
  of design problem of struts1 framework that has been resolved in the struts 2
  framework. Most of the Struts 2 classes are based on interfaces and most of its
  core interfaces are HTTP independent. Struts 2 Action classes are framework
  independent and are simplified to look as simple POJOs. Framework components
  are tried to keep loosely coupled.

- **Simplified Actions** - Actions are simple POJOs. Any java class with execute()
  method can be used as an Action class. Even we don't need to implement
  interfaces always. Inversion of Control is introduced while developing the action
  classes. This make the actions to be neutral to the underlying framework.

- **No more ActionForms** - ActionForms feature is no more known to the struts2
  framework. Simple JavaBean flavored actions are used to put properties directly.
  No need to use all String properties.

- **Simplified testability** - Struts 2 Actions are HTTP independent and framework
  neutral. This enables to test struts applications very easily without resorting to
  mock objects.

- **Intelligent Defaults** - Most configuration elements have a default value which
  can be set according to the need. Even there are xml-based default configuration
  files that can be overridden according to the need.

- **Improved results** - Unlike ActionForwards, Struts 2 Results provide flexibility
  to create multiple type of outputs and in actual it helps to prepare the response.

- **Better Tag features** - Struts 2 tags enables to add style sheet-driven markup
  capabilities, so that we can create consistent pages with less code. Struts 2 tags are
  more capable and result oriented. Struts 2 tag markup can be altered by changing
  an underlying stylesheet. Individual tag markup can be changed by editing a
  FreeMarker template. Both JSP and FreeMarker tags are fully supported.
• **Annotations introduced**: Applications in struts 2 can use Java 5 annotations as an alternative to XML and Java properties configuration. Annotations minimize the use of XML.

• **Stateful Checkboxes**: Struts 2 checkboxes do not require special handling for false values.

• **QuickStart**: Many changes can be made on the fly without restarting a web container.

• **Customizing controller**: Struts 1 lets to customize the request processor per module, Struts 2 lets to customize the request handling per action, if desired.

• **Easy Spring integration**: Struts 2 Actions are Spring-aware. Just need to add Spring beans!

• **Easy plugins**: Struts 2 extensions can be added by dropping in a JAR. No manual configuration is required!

• **AJAX support**: The AJAX theme gives interactive applications a significant boost. The framework provides a set of tags to help you ajaxify your applications, even on Dojo. The AJAX features include:
  1. AJAX Client Side Validation
  2. Remote form submission support (works with the submit tag as well)
  3. An advanced div template that provides dynamic reloading of partial HTML
  4. An advanced template that provides the ability to load and evaluate JavaScript remotely
  5. An AJAX-only tabbed Panel implementation
  6. A rich pub-sub event model
  7. Interactive auto complete tag

**Struts 1.x Vs Struts 2.x**

In the following section, we are going to compare the various features between the two frameworks. Struts 2.x is very simple as compared to struts 1.x, few of its excellent features are:

1. **Servlet Dependency**:

Actions in Struts1 have dependencies on the servlet API since the `HttpServletRequest` and `HttpServletResponse` objects are passed to the `execute` method when an Action is invoked but in case of Struts 2, Actions are not container dependent because they are made simple POJOs. In struts 2, the servlet contexts are represented as simple Maps which allows actions to be tested in isolation. Struts 2 Actions can access the original request and response, if required. However, other architectural elements reduce or eliminate the need to access the `HttpServletRequest` or `HttpServletResponse` directly.

2. **Action classes**
Programming the abstract classes instead of interfaces is one of design issues of struts1 framework that has been resolved in the struts 2 framework. Struts1 Action classes needs to extend framework dependent abstract base class. But in case of Struts 2 Action class may or may not implement interfaces to enable optional and custom services. In case of Struts 2, Actions are not container dependent because they are made simple POJOs. Struts 2 provides a base ActionSupport class to implement commonly used interfaces. Albeit, the Action interface is not required. Any POJO object with an `execute` signature can be used as an Struts 2 Action object.

3. Validation

Struts1 and Struts 2 both supports the manual validation via a `validate` method. Struts1 uses validate method on the ActionForm, or validates through an extension to the Commons Validator. However, Struts 2 supports manual validation via the `validate` method and the XWork Validation framework. The Xwork Validation Framework supports chaining validation into sub-properties using the validations defined for the properties class type and the validation context.

4. Threading Model

In Struts1, Action resources must be thread-safe or synchronized. So Actions are singletons and thread-safe, there should only be one instance of a class to handle all requests for that Action. The singleton strategy places restrictions on what can be done with Struts1 Actions and requires extra care to develop. However in case of Struts 2, Action objects are instantiated for each request, so there are no thread-safety issues. (In practice, servlet containers generate many throw-away objects per request, and one more object does not impose a performance penalty or impact garbage collection.)

5. Testability

Testing Struts1 applications are a bit complex. A major hurdle to test Struts1 Actions is that the `execute` method because it exposes the Servlet API. A third-party extension, Struts TestCase, offers a set of mock object for Struts1. But the Struts 2 Actions can be tested by instantiating the Action, setting properties and invoking methods. Dependency Injection support also makes testing simpler. Actions in struts2 are simple POJOs and are framework independent, hence testability is quite easy in struts2.

6. Harvesting Input

Struts1 uses an ActionForm object to capture input. And all ActionForms needs to extend a framework dependent base class. JavaBeans cannot be used as ActionForms, so the developers have to create redundant classes to capture input. However Struts 2 uses Action properties (as input properties independent of underlying framework) that eliminates the need for a second input object, hence reduces redundancy. Additionally in struts2, Action properties can be accessed from the web page via the
taglibs. Struts 2 also supports the ActionForm pattern, as well as POJO form objects and POJO Actions. Even rich object types, including business or domain objects, can be used as input/output objects.

7. Expression Language

Struts1 integrates with JSTL, so it uses the JSTL-EL. The struts1 EL has basic object graph traversal, but relatively weak collection and indexed property support. Struts 2 can also use JSTL, however it supports a more powerful and flexible expression language called "Object Graph Notation Language" (OGNL).

8. Binding values into views

In the view section, Struts1 uses the standard JSP mechanism to bind objects (processed from the model section) into the page context to access. However Struts 2 uses a "ValueStack" technology so that the taglibs can access values without coupling your view to the object type it is rendering. The ValueStack strategy allows the reuse of views across a range of types which may have the same property name but different property types.

9. Type Conversion

Usually, Struts1 ActionForm properties are all Strings. Struts1 uses Commons-Beanutils for type conversion. These type converters are per-class and not configurable per instance. However Struts 2 uses OGNL for type conversion. The framework includes converters for basic and common object types and primitives.

10. Control Of Action Execution

Struts1 supports separate Request Processor (lifecycles) for each module, but all the Actions in a module must share the same lifecycle. However Struts 2 supports creating different lifecycles on a per Action basis via Interceptor Stacks. Custom stacks can be created and used with different Actions as needed.

Download Struts 2.0

In this section we will download and install the Struts 2.0 on the latest version of Tomcat container. We will first download tomcat and configure it as our development server. Then we will download Struts 2.0 and install the struts-blank application on the tomcat server to check the examples that come with the struts-blank application.

Downloading Struts 2.0

Visit the Struts download site http://struts.apache.org/download.cgi and download the Struts 2.0 for this tutorial.
We have downloaded the Struts 2.0.6 (struts-2.0.6-all.zip) for this tutorial.

**Download the Tomcat**

Download the latest version of tomcat from [http://tomcat.apache.org/download-55.cgi](http://tomcat.apache.org/download-55.cgi). We have downloaded **apache-tomcat-5.5.20.zip** for this tutorial.
I am assuming that latest version of Java is installed on your system. Extract downloaded file and run the C:\apache-tomcat-5.5.20\bin\startup.bat to start the tomcat. Type http://localhost:8080/ in your browser, it should show the welcome page in the browser window as shown below.

Please see the README file for packaging information. It every distribution contains.

### Binary Distributions

- **Core:**
  - zip (pgp, md5)
  - tar.gz (pgp, md5)
  - Windows Service Installer (pgp, md5)

- **Deployer:**
  - zip (pgp, md5)
  - tar.gz (pgp, md5)

- **Embedded:**
  - zip (pgp, md5)
  - tar.gz (pgp, md5)

- **Administration Web Application:**
  - zip (pgp, md5)
  - tar.gz (pgp, md5)

- **JDK 1.4 Compatibility Package:**
  - zip (pgp, md5)
  - tar.gz (pgp, md5)

- **Documentation (Already Included in Core Downloads):**
  - tar.gz (pgp, md5)
Congratulations you have now successfully installed latest version of tomcat for learning Struts 2.0. Now we will install the struts-blank application on the tomcat container and test the application.

**Extracting Struts 2.0 distribution and installing on tomcat**

Extract the downloaded struts distribution struts-2.0.6-all.zip into your favorite directory. To install the struts blank application copy "struts2-blank-2.0.6" from 
"<extracted directory>\struts-2.0.6-all\struts-2.0.6\apps" into the webapps directory of tomcat. Tomcat will automatically deploy the application.
To test the struts-blank application type *http://localhost:8080/struts2-blank-2.0.6* in the browser and the struts-blank application get displayed in your browser window.

Struts is up and running ...

Languages

- English
- Español

Click the English link and familiarize yourself with Struts Blank sample application.

Struts 2.0 distribution also contains the following sample applications that you can try:

- struts2-mailreader-2.0.6.war
- struts2-portlet-2.0.6.war
- struts2-showcase-2.0.6.war

You can simply copy these files to the webapps directory of your tomcat server. Tomcat will automatically deploy these applications and then you can test these applications.
Struts 2 Hello World - Developing Hello World Application

In this section we will develop Hello World application based on Struts 2 Framework. Our Struts 2 Hello World application is your first step towards developing applications based on Struts 2 Framework. We are here providing step by step guide for developing Hello World application in Struts 2 framework.

Tutorial covers basic steps like creating directory structure, developing build.xml file to build the application using ant build tool. Then we explained Java, JSP and configuration files that are required in this application.

Creating directory structure for the project

**Step1:** Extract struts 2 download and copy struts2-blank-2.0.6.war (If you are using latest version of struts 2 then version may be different for you) to your tomcats webapps directory. Rename struts2-blank-2.0.6.war to struts2tutorial and unzip it in the tomcats webapps directory. Now start the tomcat and type [http://localhost:8080/struts2tutorial/](http://localhost:8080/struts2tutorial/) into your browser. You browser should show look like:
Congratulations you have successfully installed struts 2 blank application to start with.

**Step 2:** Now delete the content of `struts2tutorial\WEB-INF\src` and `struts2tutorial\WEB-INF\classes` directories, as we don't need these files that comes with struts 2 blank application.

**Step 3:** Create `build.xml` file in the `struts2tutorial\WEB-INF\src` and paste the following content in the `build.xml` file.

```xml
<project name="Struts 2 Tutorial" basedir="." default="all">
  <!-- Project settings -->
  <property name="project.title" value="RoseIndia Struts 2 Tutorials"/>
  <property name="project.jar.file" value="struts2tutorial.jar"/>

  <path id="class.path">
    <fileset dir="lib">
      <include name="**/*.jar"/>
    </fileset>
    <fileset dir="libext">
      <include name="**/*.jar"/>
    </fileset>
  </path>

  <!-- Classpath for Project -->
  <path id="compile.classpath">
    <pathelement path ="lib/commons-beanutils.jar"/>
    <pathelement path ="lib/commons-digester.jar"/>
    <pathelement path ="lib/struts.jar"/>
    <pathelement path ="libext/servlet-api.jar"/>
    <pathelement path ="libext/catalina-ant.jar"/>
    <pathelement path ="classes"/>
    <pathelement path ="${classpath}"/>
  </path>

  <!-- Check timestamp on files -->
  <target name="prepare">
  </target>
</project>
```
<copy>
  file="src/struts.xml"
  todir="src/classes"/>
</copy>

<!-- Copy any resource or configuration files -->

<target name="resources">
  <copy todir="src/classes" includeEmptyDirs="no">
    <fileset dir="src/java">
      <patternset>
        <include name="**/*.conf"/>
        <include name="**/*.properties"/>
        <include name="**/*.xml"/>
      </patternset>
    </fileset>
  </copy>
</target>

<!-- Normal build of application -->

<target name="compile" depends="prepare,resources">
  <javac srcdir="src" destdir="src/classes"
        debug="true" debuglevel="lines,vars,source">
    <classpath refid="class.path"/>
  </javac>
  <jar jarfile="lib/${project.jar.file}"
       basedir="src/classes"/>
</target>

<!-- Remove classes directory for clean build -->

<target name="clean"
        description="Prepare for clean build">
  <delete dir="classes"/>
</target>
<mkdir dir="classes"/>
</target>

<!-- Build Javadoc documentation -->
<target name="javadoc"

description="Generate JavaDoc API docs">
  <delete dir="./doc/api"/>
  <mkdir dir="./doc/api"/>
  <javadoc sourcepath="./src/java"
    destdir="./doc/api"
    classpath="${servlet.jar}:${jdbc20ext.jar}"
    packagenames="*"
    author="true"
    private="true"
    version="true"
    windowtitle="${project.title} API Documentation"
    doctitle="&lt;h1&gt;${project.title} Documentation (Version ${project.version})&lt;/h1&gt;"
    bottom="Copyright © 2002">
    <classpath refid="compile.classpath"/>
  </javadoc>
</target>

<!-- Build entire project -->
<target name="project" depends="clean,prepare,compile"/>

<!-- Create binary distribution -->
<target name="dist"

description="Create binary distribution">
  <mkdir dir="bin"/>
  ...
</target>
Step 4: Create directory **libext** under the **struts2tutorial\WEB-INF\** then you copy latest Servlets api jar (in our case **servlet-api.jar**) file over there. This library file will be used to compile Servlets in our application.

Step 5: Now create directories **java** and **classes** under **struts2tutorial\WEB-INF\src**. The directory **struts2tutorial\WEB-INF\src\java** will be used to put all the java sources file and directory **struts2tutorial\WEB-INF\src\classes** will be used by ant build utility to store compiled java files.

Now we have successfully created the directory structure and ant build file for our Struts 2 Hello World Application. In the next section we will create **JSP**, Java file and the configuration file and then test our Struts 2 Hello World application.
Writing JSP, Java and Configuration for Hello World Application

In this section we will write JSP, Java and required configuration files for our Struts 2 Hello World application. Now in struts 2 struts.xml is used to configure the applications.

Understanding the application

Our application is very simple application that displays Hello World message along with current date and time of the server. When user clicks on the "Run Struts 2 Hello World Application" link on the tutorial home page, a request is sent to the struts framework. Then struts framework sends the input to the action class (in our case Struts2HelloWorld.java). After action is fired the Result selects the resource "/pages/HelloWorld.jsp" to render the response.

In this example we have to develop three parts view, Action class and mapping (struts.xml) to couple action and page. By creating these three components we are separating the application in three parts View, Model and Controller.

Developing View:
This page is used to display the result on the browser. The HelloWorld.jsp is view part of our application. Create "HelloWorld.jsp" in the struts2tutorial\pages directory and add the following content:

```html
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
<head>
<title>Struts 2 Hello World Application!</title>
</head>
<body>
<h2><s:property value="message" /></h2>
<p>Current date and time is: <b><s:property value="currentTime" /></b></p>
</body>
</html>
```

The line `<%@ taglib prefix="s" uri="/struts-tags" %>` declares data tag library of struts. The struts data tag is used to display the dynamic data. The tag `<s:property value="message" />` and `<s:property value="currentTime" />` calls the methods getMessage() and getCurrentTime() respectively of the Struts2HelloWorld action class and merges the values with response.

Developing Action (to interact with Model):
Now create `Struts2HelloWorld.java` and saves it to the "struts2tutorial\WEB-INF\src\java\net\roseindia" directory. This action class creates the message to be displayed on the screen. Here is the code of `Struts2HelloWorld.java`:

```java
package net.roseindia;
import com.opensymphony.xwork2.ActionSupport;
import java.util.Date;

public class Struts2HelloWorld extends ActionSupport {
    public static final String MESSAGE = "Struts 2 Hello World Tutorial! ";

    public String execute() throws Exception {
        setMessage(MESSAGE);
        return SUCCESS;
    }

    private String message;

    public void setMessage(String message){
        this.message = message;
    }

    public String getMessage() {
        return message;
    }

    public String getCurrentTime(){
        return new Date().toString();
    }
}
```

**Developing Controller Configuration File:**

Struts 2 uses the `struts.xml` file for configuring the application. Create `struts.xml` file and save it in the "struts2tutorial\WEB-INF\src" directory with the following content.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN" "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>

<constant name="struts.enable.DynamicMethodInvocation" value="false" />
<constant name="struts.devMode" value="true" />

<package name="roseindia" namespace="/roseindia" extends="struts-default"/>
</struts>
```
The struts.xml file should be present in the class path of the application, you can either include it in the jar and place in the lib directory of the application or place it in the classes directory of the web application. In our application we are using ant build tool which is including it in the jar file.

**Building the application**

I am assuming that you have already installed ant build tool on your machine. Since we are using the ant built tool, building application by using is very easy. To build the application open command prompt and go to "struts2tutorial\WEB-INF\src" directory of the web application and issue the "ant" command. The ant build tool will compile the java file and create jar file "struts2tutorial.jar" into the lib directory of your web application. Here is the output of ant build tool:

```
C:\apache-tomcat-6.0.10\apache-tomcat-6.0.10\webapps\struts2tutorial\WEB-INF\src
>ant
Buildfile: build.xml

clean:
[delete] Deleting directory C:\apache-tomcat-6.0.10\apache-tomcat-6.0.10\webapps\struts2tutorial\WEB-INF\classes
[mkdir] Created dir: C:\apache-tomcat-6.0.10\apache-tomcat-6.0.10\webapps\struts2tutorial\WEB-INF\classes

prepare:

resources:

compile:
[javac] Compiling 1 source file to C:\apache-tomcat-6.0.10\apache-tomcat-6.0.10\webapps\struts2tutorial\WEB-INF\src\classes

[jar] Building jar: C:\apache-tomcat-6.0.10\apache-tomcat-6.0.10\webapps\struts2tutorial\WEB-INF\lib\struts2tutorial.jar
```
Testing Struts 2 Hello World Application

In the above section we have compiled our application and now finally we will test our application. To test the application start the tomcat server and type http://localhost:8080/struts2tutorial/ and then select "Run Struts 2 Hello World Application" from the list.

Here is the screen shot of our struts 2 tutorial home page:

Select "Run Struts 2 Hello World Application" link. Our first application "Struts 2 Hello World" will look like the following:
The application will display message "Struts 2 Hello World Tutorial!" along with current date and time of the server.

**How application works?**

Here is the brief description on how Struts 2 Hello World Application works:

Your browser sends a request to the web server for the URL http://localhost:8080/tutorial/HelloWorld.action.

1. When you click on the **Run Struts 2 Hello World Application** link, the browser sends a request for the url http://localhost:8080/struts2tutorial/roseindia/HelloWorld.action. The container requests for the resource "HelloWorld.action". By default web.xml file of struts blank application is configured to route all the request for *.action through org.apache.struts2.dispatcher.FilterDispatcher. Here is the configuration from web.xml file:

```xml
<filter>
    <filter-name>struts2</filter-name>
    <filter-class>org.apache.struts2.dispatcher.FilterDispatcher</filter-class>
</filter>

<filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

2. Then the framework looks for the mapping for the action "HelloWorld" and then framework instantiates the appropriate class and calls the execute method. In this case action class is Struts2HelloWorld. Here is the configuration file from struts.xml which defines the action mapping:

```xml
<action name="HelloWorld" class="net.roseindia.Struts2HelloWorld">
    <result>/pages/HelloWorld.jsp</result>
</action>
```
3. Then the execute method sets the message and returns SUCCESS.
   
   ```java
   public String execute() throws Exception {
       setMessage(MESSAGE);
       return SUCCESS;
   }
   ```

   Then framework determines which page is to be loaded if SUCCESS is returned. In our case framework tells the container to load HelloWorld.jsp and render the output.

   In the struts 2 framework Actions are used to process the form and user request. The execute method of the action returns SUCCESS, ERROR, or INPUT value. Then based on these values framework tells the container to load and render the appropriate result.

   4. Container process the HelloWorld.jsp and generates the output.

   5. Then the output in the HTML format is sent to the browser.

**Download the application and source code:**
You can download the application and source code of the tutorial and to save your time and efforts spent on configuring the application. [Click here to download the code.](#)

Our goal is to provide the working skeleton of the application based on ant tool. So you can download the application and then use ant tool to compile the java classes. Application is in the exploded directory structure and can be deployed as it is on tomcat server. If any of our visitor finds and problem please discuss it. We request you to share your experiences with us.

### Struts Configuration file - struts.xml

In this section we will introduce you to the struts.xml file. This section explains you how best you can use the struts.xml file for you big projects.

**The struts.xml File**

The Struts 2 Framework uses a configuration file (struts.xml) to initialize its own resources. These resources include:

- Interceptors that can preprocess and postprocess a request
- Action classes that can call business logic and data [access code](#)
- Results that can prepare views using JavaServer Pages, Velocity and FreeMarker templates
At runtime, there is a single configuration for an application. Prior to runtime, the configuration is defined through one or more XML documents, including the default struts.xml document. There are several elements that can be configured, including packages, namespaces, includes, actions, results, interceptors, and exceptions.

The struts.xml file is the core configuration file for the framework and it should be present in the class path of your web application. Features of struts 2 configuration file:

- The struts.xml file allows to break big struts.xml file into small files and configuration files to be included as needed. Here is the example:

```xml
<struts>
    ....
    
    <include file="file1.xml"/>
    <include file="file2.xml"/>
    ....
    
</struts>
```

- You can even place struts-plugin.xml file in the JAR, and it will be automatically plugged into the application. This helps the programmers to develop self-configured components.

- If you want to use the frameworks such as Freemaker and Velocity modules, then the templates can also be loaded from classpath. This enables the developer to package entire module just in single JAR file.

**Structure of the struts.xml file**

In the last section we developed and tested the Hello World application. Here is the sample struts.xml file from the last example.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
```
The struts.xml file must confirm to the Struts 2 Document Type Definition (DTD)

The DTD provides information about the structure and the elements that the struts.xml file should have.
Here is the Struts 2.0 DTD:

<!DOCTYPE struts PUBLIC
  "-//Apache Software Foundation//DTD Struts
  Configuration 2.0//EN"
  "http://struts.apache.org/dtds/struts-2.0.dtd">

<!ELEMENT struts (package|include|bean|constant)>
<!ELEMENT package (result-types?, interceptors?,
default-interceptor-ref?, default-action-ref?, global-
results?,
global-exception-mappings?, action*)>
<!ATTLIST package
  name CDATA #REQUIRED
  extends CDATA #IMPLIED
  namespace CDATA #IMPLIED
  abstract CDATA #IMPLIED
  externalReferenceResolver NMTOKEN #IMPLIED
It is possible to remove the “struts.xml” file from your application completely if the functionality of your application does not depend on it. There are few configurations that can be handled alternatively such as annotations, “web.xml” startup parameters, and alternate URL mapping schemes. Still, there are few configurations that always need the “struts.xml” file like the global results, exception handling, and the custom interceptor stacks.

Exploring struts.xml

The <struts> tag is the root tag for the struts.xml. It may contain the following tags: package, include, bean and constant.

1. The Package Tag:

Packages are a way to group actions, results, result types, interceptors, and interceptor-stacks into a logical configuration unit. Conceptually, packages are similar to objects in that they can be extended and have individual parts that can be overridden by "sub" packages.
The `<package />` tag is used to group together configurations that share common attributes such as interceptor stacks or URL namespaces. It may also be useful to organizationally separate functions, which may be further separated into different configuration files.

The package element has one required attribute, `name`, which acts as the key for later reference to the package. The `extends` attribute is optional and allows one package to inherit the configuration of one or more previous packages - including all interceptor, interceptor-stack, and action configurations.

Note that the configuration file is processed sequentially down the document, so the package referenced by an "extends" should be defined above the package which extends it.

The optional `abstract` attribute creates a base package that can omit the action configuration.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>yes</td>
<td>key for other packages to reference</td>
</tr>
<tr>
<td>extends</td>
<td>no</td>
<td>inherits package behavior of the package it extends</td>
</tr>
<tr>
<td>namespace</td>
<td>no</td>
<td>provides a mapping from the URL to the package.</td>
</tr>
<tr>
<td>abstract</td>
<td>no</td>
<td>declares package to be abstract (no action configurations required in package)</td>
</tr>
</tbody>
</table>

1. **name** – unique name is given for a package.
2. **extends** – the name of a package that this package will extend; all configuration information (including action configurations) from the extended package will be available in the new package, under the new namespace.
3. **namespace** – the namespace provides a mapping from the URL to the package. i.e. for two different packages, with namespace attributes defined as “pack1” and “pack2”, the URLs would be something like “/webApp/pack1/my.action” and “/webApp/pack2/my.action”.
4. **abstract** – if this attribute value is “true” the package is truly a configuration grouping and actions configured will not be accessible via the package name. It is important to make sure you are extending the correct parent package so that the necessary pre-configured features will be available to you.

2. **The Include Tag:**

The `<include />` tag is used to modularize a Struts2 application that needs to include other configuration files. It contains only one attribute “file” that provides the name of the xml file to be included. This file has exactly the same structure as the “struts.xml” configuration file. For example, to break a configuration file of a finance application, you might choose to group together the invoices, admin, report configurations etc into separate files:
While including files, order is important. The information from the included file will be available from the point that the include tag is placed in the file.

There are some files that are included implicitly. These are the “strutsdefault.xml” and the “struts-plugin.xml” files. Both contains default configurations for result types, interceptors, interceptor stacks, packages as well as configuration information for the web application execution environment (which can also configured in the “struts.properties” file). The difference is that “struts-default.xml” provides the core configuration for Struts2, where “struts-plugin.xml” provides configurations for a particular plug-in. Each plug-in JAR file should contain a “struts-plugin.xml” file, all of which are loaded during startup.

3. The Bean Tag

Most applications won't need to extend the Bean Configuration. The bean element requires the class attribute which specifies the Java class to be created or manipulated. A bean can either

1. be created by the framework's container and injected into internal framework objects, or
2. have values injected to its static methods

The first use, object injection, is generally accompanied by the type attribute, which tells the container that which interface this object implements.

The second use, value injection, is good for allowing objects not created by the container to receive framework constants. Objects using value inject must define the the static attribute.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>class</td>
<td>yes</td>
<td>the name of the bean class</td>
</tr>
<tr>
<td>type</td>
<td>no</td>
<td>the primary Java interface this class implements</td>
</tr>
<tr>
<td>name</td>
<td>no</td>
<td>the unique name of this bean; must be unique among other</td>
</tr>
</tbody>
</table>
beans that specify the same type
the scope of the bean; must be either default, singleton, request, session, thread
whether to inject static methods or not; shouldn't be true when the type is specified
whether the bean is optional or not

**Bean Example (struts.xml)**

```xml
<struts>
  <bean
type="roseindia.net.ObjectFactory"
name="factory"
class="roseindia.net.MyObjectFactory"/>
  ...
</struts>
```

4. The **Constant Tag**

There are two key roles for constants.

1. They are used to override settings like the maximum file upload size or whether the Struts framework should be in devMode(= development mode) or not.
2. They specify which Bean should be chosen, among multiple implementations of a given type.

Constants can be declared in multiple files. By default, constants are searched for in the following order, allowing for subsequent files to override by the previous ones:

- struts-default.xml
- struts-plugin.xml
- struts.xml
- struts.properties
- web.xml

The struts.properties file is provided for backward-compatibility with WebWork. In the struts.properties file, each entry is treated as a constant. In the web.xml file, any FilterDispatcher initialization parameters are loaded as constants.
In the various XML variants, the constant element has two required attributes: `name` and `value`.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>yes</td>
<td>the name of the constant</td>
</tr>
<tr>
<td>value</td>
<td>yes</td>
<td>the value of the constant</td>
</tr>
</tbody>
</table>

**Constant Example (struts.xml)**

```xml
<struts>
  <constant name="struts.devMode" value="true"/>
  ...
</struts>
```

**Constant Example (struts.properties)**

```
struts.devMode = true
```

**Struts 2 Login Application**

**Developing Struts 2 Login Application**

In this section we are going to develop login application based on Struts 2 Framework. Our current login application does not validate the user against the database. Instead login name and password are validated against the hardcode values (User: Admin and Password: Admin) in the actions class.

**Working of the application**

1. Login page is displayed to take the input.
2. User enters user name and password and then clicks on the "Login" button.
3. User validation is done in action class and if user enters Admin/Admin in the user name/password fields, then success pages is displayed. Otherwise the error message is displayed on the screen.

**Steps to develope the application**

Here are simple and easy steps to develop Login page in the using Struts 2 framework.
1. **Develop Login Form**

   The GUI of the application consists of login form (login.jsp) and success message page (loginsuccess.jsp).

   The login.jsp is used to display the login page to the user. In our application it is saved in "webapps\struts2tutorial\pages" folder. Here is the code of login.jsp file:

   ```jsp
   <%@ taglib prefix="s" uri="/struts-tags" %>
   <html>
   <head>
   <title>Struts 2 Login Application!</title>
   <link href="/css/main.css" rel="stylesheet" type="text/css"/>
   </head>
   <body>
   <s:form action="doLogin" method="POST">
   <tr>
   <td colspan="2">
   Login
   </td>
   </tr>
   <tr>
   <td colspan="2">
   <s:actionerror />
   <s:fielderror />
   </td>
   </tr>
   <s:textfield name="username" label="Login name"/>
   <s:password name="password" label="Password"/>
   <s:submit value="Login" align="center"/>
   </s:form>
   </body>
   </html>
   ```

2. The code:

   
   ```jsp
   <s:actionerror />
   <s:fielderror />
   ```

   displays action errors and field validation errors.

3. The code `<s:form action="doLogin" method="POST">` generates the html form for the application.

   The code:
generates Login Name and Password fields.

4. The submit button is generated through `<s:submit value="Login" align="center"/>` code.

When application is executed it generates the following HTML code:

```html
<html>
<head>
    <title>Struts 2 Login Application!</title>
    <link href="/struts2tutorial/css/main.css" rel="stylesheet" type="text/css"/>
</head>
<body>
<form id="doLogin" name="doLogin" onsubmit="return true;"
action="/struts2tutorial/roseindia/doLogin.action" method="POST">
<table class="wwFormTable">
<tr>
    <td colspan="2">Login</td>
</tr>
<tr>
    <td class="tdLabel"><label for="doLogin_username" class="label">Login name:</label></td>
    <td><input type="text" name="username" value="" id="doLogin_username"/>
</td>
</tr>
<tr>
    <td class="tdLabel"><label for="doLogin_password" class="label">Password:</label></td>
    <td><input type="password" name="password" id="doLogin_password"/>
</td>
</tr>
<tr>
    <td colspan="2"><div align="center"><input type="submit" id="doLogin_0" value="Login"/></div></td>
</tr>
</table></form>
</body>
</html>
```
5. On viewing the above generated html code you will find that Struts 2 automatically generates form, html table, label for the html elements. This is the another great feature of Struts as compared to Struts 1.x.

The loginsuccess.jsp page displays the Login Success message when user is authenticated successfully. Here is the code of loginsuccess.jsp file:

```html
<html>
<head>
<title>Login Success</title>
</head>
<body>
<p align="center"><font color="#000080" size="5">Login Successful</font></p>
</body>
</html>
```

6. **Developing Action Class**

Now let's develop the action class to handle the login request. In Struts 2 it is not necessary to implement the Action interface, any POJO object with execute signature can be used in Struts 2. The Struts 2 framework provides a base ActionSupport class to implement commonly used interfaces. In our action class (Login.java) we have implemented ActionSupport interface. Our "Login.java" is saved in the "webapps\struts2tutorial\WEB-INF\src\java\net\roseindia" directoy. Here is the code of Login.java action class:

```java
package net.roseindia;
import com.opensymphony.xwork2.ActionSupport;
import java.util.Date;

/**
 * <p> Validate a user login. </p>
 */
public class Login extends ActionSupport {
```
```java
public String execute() throws Exception {
    System.out.println("Validating login");
    if (!getUsername().equals("Admin") || !
        getPassword().equals("Admin")){
        addActionError("Invalid user name or password! Please try again!");
        return ERROR;
    }else{
        return SUCCESS;
    }
}

// ---- Username property ----

/**
 * Field to store User username.
 * @return Returns the User username.
 */
private String username = null;

/**
 * Provide User username.
 * @param value The username to set.
 */
public String getUsername() {
    return username;
}

/**
 * Store new User username
 * @param value The username to set.
 */
public void setUsername(String value) {
    username = value;
}

// ---- Username property ----

/**
 * Field to store User password.
 * @return Returns the User password.
 */
private String password = null;

/**
 * Provide User password.
 * @return Returns the User password.
 */
public String getPassword() {
```
8. **Configuring action mapping (in struts.xml)**

Now we will create action mapping in the struts.xml file. Here is the code to be added in the struts.xml:

```xml
<action name="showLogin">
    <result>/pages/login.jsp</result>
</action>

<action name="doLogin" class="net.roseindia.Login">
    <result name="input">/pages/login.jsp</result>
    <result name="error">/pages/login.jsp</result>
    <result>/pages/loginsuccess.jsp</result>
</action>
```

10. In the above mapping the action "showLogin" is used to display the login page and "doLogin" validates the user using action class (Login.java).

11. **CSS file (main.css)**

This css file is used to enhance the presentation of the login form. The **main.css** is saved into "\webapps\struts2tutorial\css" directory.

Here is the code of **main.css**:

```css
@CHARSET "UTF-8";

body {
    font: 12px verdana, arial, helvetica, sans-serif;
    background-color:#FFFFFF;
}

table.wwFormTable {
    font: 12px verdana, arial, helvetica, sans-serif;
    border-width: 1px;
    border-color: #030;
    border-style: solid;
    color: #242;
    background-color: #ada;
}
```
In the next section we will learn how to add validation to the login application.

**Struts 2 Validation Example**

**Validation Login application**

In this section we will write the code to validate the login application. After completing this section you will be able to write validations for your Struts 2 projects. The concepts defined in this section are so illustrative that a learner quickly develops his/her skills in Struts 2 framework.
Struts 2 is very elegant framework that provides a lot of functionality to develop web based applications quickly. Here you will learn to write the form validation code in Struts 2 very easily. We will add the form validation code in our login application.

For validation the login application javascript can be added to the jsp page or in action class, but Struts 2 provides another very easy method to validate your fields automatically. You can even use the same configuration file to generate client side script (in next section we will see how to generate client side validation code).

The Struts 2 validation framework uses xml based configuration file. The file name should be <Your action class> -validation.xml. In our case our action class name is Login.java, so our validation configuration file will be Login-validation.xml. The Login-validation.xml will be saved into "webapps\struts2tutorial\WEB-INF\src\java\net\roseindia" directory. Here is the content of Login-validation.xml file:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE validators PUBLIC
    "-//OpenSymphony Group//XWork Validator 1.0.2//EN"
    "http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">

<validators>
  <field name="username">
    <field-validator type="requiredstring">
      <param name="trim">true</param>
      <message>Login name is required</message>
    </field-validator>
  </field>
  <field name="password">
    <field-validator type="requiredstring">
      <param name="trim">true</param>
      <message>Password is required</message>
    </field-validator>
  </field>
</validators>
```
In the above configuration file the field name corresponds to the ActionForm properties. For the `username` and `password` elements the `requiredstring` validation is applied and the message in the `<message>...</message>` tag is used to display the message if validation fails.

**Compiling the application**

To compile the application go to "\webapps\struts2tutorial\WEB-INF\src" directory and type ant command. The ant tool will compile the application for you.

**Adding the link into index.html**

Finally we add the link in the index.html to access the login form.

```html
<ul>
<li><a href="roseindia/showLogin.action">Login Application</a></li>
</ul>
```

In the next section we will run and test the application.

**Running and Testing Struts 2 Login application**

**Running Struts 2 Login Example**
In this section we will run the example on Tomcat 6.0 server and check how it works.

**Running Tomcat**

To run the Tomcat, go to its bin directory and then double click on startup.bat. The startup.bat will start the tomcat.

**Testing application**

To test the application type `http://localhost:8080/struts2tutorial/`. Your browser should show the following screen:
Now click on "Login Application" link. Then your browser will display the Login page as shown below:

Now click on the "Login" button. Application will show the following error screen:

Now you enter the Login name "Admin" and click on the "Login" button, application will show the following error:
Now enter any password except "Admin", application will again show the error. This error is actually generated in the Action class.

Now enter the valid password "Admin" and click on "Login" button. Now the application will show you the welcome message as shown below.

**Login Successful**

Congratulations! Now you have successfully developed and tested your Struts 2 Login application. In the next section we will show you how to add client side JavaScript validation using Struts 2 Validation framework.

**Client Side validation in Struts 2 application**

In this section we will see how to write code that will generate JavaScript code for client side validation. In the last section we developed Login-validator.xml configuration file.
for defining the server side validation. In this section we will use the same Login-validator.xml file for generating the client side java script.

Developing JSP pages

Here is the code of login jsp page (loginClientSideValidation.jsp)

```jsp
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
<head>
<title>Struts 2 Login Application!</title>
<link href="<s:url value="/css/main.css"/>" rel="stylesheet" type="text/css"/>
</head>
<body>

<s:form action="doLoginClientSideValidation" method="POST" validate="true">
<tr>
<td colspan="2">
Login</td>
</tr>
<s:actionerror /> 
<s:fielderror />
<s:textfield name="username" label="Login name"/>
<s:password name="password" label="Password"/>
<s:submit value="Login" align="center"/>
</s:form>
</body>
</html>
```

Note that in the above code we have just added validate="true" in the <s:form tag.../>. This is the only work we have to do and rest work is done by Struts 2 validator.
framework. The validator framework generates JavaScript for validating the form at client side.

**Changes in the struts.xml**

Add the following code into struts.xml file:

```xml
<action name="showLoginClientSideValidation">
  <result>/pages/loginClientSideValidation.jsp</result>
</action>

<action name="doLoginClientSideValidation" class="net.roseindia.Login">
  <result name="input">/pages/loginClientSideValidation.jsp</result>
  <result name="error">/pages/loginClientSideValidation.jsp</result>
  <result>/pages/loginsuccess.jsp</result>
</action>
```

The action **showLoginClientSideValidation** displays login form while doLoginClientSideValidation handles the validation request.

**Adding the link into index.html**

Finally we add the link in the index.html to access the login form for testing client side validation.

```html
<ul>
  <li><a href="roseindia/showLoginClientSideValidation.action">Login Application (Client Side Validation)</a></li>
</ul>
```

**Testing the Client side validation**

Start tomcat and type [http://localhost:8080/struts2tutorial/](http://localhost:8080/struts2tutorial/). Your browser should show the following screen:
Now click on "Login Application (Client Side Validation)" link. Then your browser will display the Login page as shown below:

```
Login
Login name: ___________________________
Password: ___________________________
         Login
```

Click on the "Login" button without entering anything. Java Script will show the error message as shown below:

```
Login
Login name is required
Login name: ___________________________
Password is required
Password: ___________________________
         Login
```

Now enter the "Login Name" and click on the "Login" button, application will show error as shown below:
Examining the Java Script code generated

The following html code is generated by the framework:

```html
<html>
<head>
<title>Struts 2 Login Application!</title>
<link href="/struts2tutorial/css/main.css" rel="stylesheet" type="text/css"/>
</head>

<body>
<script src="/struts2tutorial/struts/xhtml/validation.js"></script>
<form namespace="/roseindia" id="doLoginClientSideValidation" name="doLoginClientSideValidation" onsubmit="return validateForm_doLoginClientSideValidation();" action="/struts2tutorial/roseindia/doLoginClientSideValidation.action" method="POST">
<table class="wwFormTable">
<tr>
<td colspan="2">
Login
</td>
</tr>
<tr>
<td class="tdLabel"><label for="doLoginClientSideValidation_username" class="label">Login name:</label></td>
<td>
Admin
</td>
</tr>
</table>
</form>
</body>
</html>
```
```html
<form id="doLoginClientSideValidation">
  <table>
    <tr>
      <td class="tdLabel"><label for="doLoginClientSideValidation_username" class="label">Username:</label></td>
      <td><input type="text" name="username" value="" id="doLoginClientSideValidation_username"/>
    </tr>
    <tr>
      <td class="tdLabel"><label for="doLoginClientSideValidation_password" class="label">Password:</label></td>
      <td><input type="password" name="password" id="doLoginClientSideValidation_password"/>
    </tr>
    <tr>
      <td colspan="2"><div align="center"><input type="submit" id="doLoginClientSideValidation_0" value="Login"/>
    </td>
  </table>
</form>

<script type="text/javascript">
  function validateForm_doLoginClientSideValidation() {
    form = document.getElementById("doLoginClientSideValidation");
    clearErrorMessages(form);
    clearErrorLabels(form);
    var errors = false;

    // field name: username
  }
</script>
```
// validator name: requiredstring

if (form.elements['username']) {
    field = form.elements['username'];
    var error = "Login name is required";
    if (field.value != null && (field.value == "" || field.value.replace(/\s+\s+/g, "").length == 0)) {
        addError(field, error);
        errors = true;
    }
}

// field name: password

// validator name: requiredstring

if (form.elements['password']) {
    field = form.elements['password'];
    var error = "Password is required";
    if (field.value != null && (field.value == "" || field.value.replace(/\s+\s+/g, "").length == 0)) {
        addError(field, error);
        errors = true;
    }
}

return !errors;

</script>

</body>

</html>
In the above code you can see the JavaScript code and function validateForm_doLoginClientSideValidation() which is generated for client side validation

**Validations using Struts 2 Annotations**

**Validating Struts 2 Login Application using Annotations**
In this section we are going to validate our login application using Annotations in Action class. Our current login application does not validate the user against the database. Instead login name and passwords are validated against the hardcode values (User: Admin and Password: Admin) in the actions class.

**Working of the application**

1. Login page is displayed to take the inputs.
2. User enters user name and password and then clicks on the "Login" button.
3. User validation is done in action class and if user enters Admin/Admin in the user name/password fields, then success pages is displayed. Otherwise the error message is displayed on the screen.

**Steps to develop the application**

Here are simple and easy steps to develop Login page in the using Struts 2 framework.

1. **Develop Login Form**
   The GUI of the application consists of login form (log-in.jsp) and success message page (loginsuccess.jsp).
   The log-in.jsp is used to display the login page to the user. In our application it is saved in "webapps\struts2tutorial\pages" folder. Here is the code of log-in.jsp file:

```html
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
<head>
<title>Struts 2 Login Application!</title>
<link href="<s:url value="/css/main.css"/>" rel="stylesheet" type="text/css"/>
</head>
<body>
<s:form action="AnnotationAction" method="POST" validate="true">
<tr>
```
2. The code:
   `<s:actionerror />`
   `<s:fielderror />`
   displays action errors and field validation errors.

3. The code `<s:form action="AnnotationAction" method="POST" validate="true">` generates the html form for the application.

   The code:
   `<s:textfield name="username" label="Login name"/>`
   `<s:password name="password" label="Password"/>`
   generates Login Name and Password fields.

4. The submit button is generated through `<s:submit value="Login" align="center"/>` code.

The `loginsuccess.jsp` page displays the Login Success message when user is authenticated successfully. Here is the code of `loginsuccess.jsp` file:

```html
<html>
<head>
```
5. Developing Action class (using Annotations to validate Login forms)
Now let's develop the action class to handle the login request. The Struts 2 framework provides a base ActionSupport class to implement commonly used interfaces. In our action class (AnnotationAction.java) we have extended ActionSupport class and imported the com.opensymphony.xwork2.validator.annotations package.
For validating the login application java script can be added to the jsp page or in action class, but Struts 2 provides another very easy method to validate your form fields using annotations in the action class.
Two annotations are needed,
1. The @Validation annotation tells to Struts that action in this class might need to be validated.
2. The @RequiredStringValidator annotation is used for the text input to hold a singular value.
Rest of the care is taken by the framework.

Our "AnnotationAction" class is saved in the "webapps\struts2tutorial\WEB-INF\src\java\net\roseindia" directoy. Here is the code of AnnotationAction.java action class:
In this class we will write the code to validate the login page.

    AnnotationAction.java

    package net.roseindia;
    import com.opensymphony.xwork2.ActionSupport;
    import com.opensymphony.xwork2.validator.annotations. *;
    @Validation
public class AnnotationAction extends ActionSupport {

private String username = null;
private String password = null;

@RequiredStringValidator(message="Supply name")
public String getUsername() {
    return username;
}

public void setUsername(String value) {
    username = value;
}

@RequiredStringValidator(message="Supply password")
public String getPassword() {
    return password;
}

public void setPassword(String value) {
    password = value;
}

public String execute() throws Exception {

    System.out.println("Validating login");
    if (!getUsername().equals("Admin") || !password.equals("Admin")) {
        addActionError("Invalid user name or password! Please try again!");
        return ERROR;
    } else {
        return SUCCESS;
    }
}
1. **Configuring action mapping (in struts.xml)**
   Now we will create action mapping in the struts.xml file. Here is the code to be added in the struts.xml:

   ```xml
   <action name="LoginAnnotation">
   <result>/pages/log-in.jsp</result>
   </action>

   <action name="AnnotationAction" class="net.roseindia.AnnotationAction">
   <result name="input">/pages/log-in.jsp</result>
   <result name="error">/pages/log-in.jsp</result>
   <result>/pages/loginsuccess.jsp</result>
   </action>
   ``

2. In the above mapping the action "LoginAnnotation" is used to display the login page and "AnnotationAction" validates the user using action class (AnnotationAction.java).

3. **CSS file (main.css)**
   This css file is used to enhance the presentation of the login form. The main.css is saved into "webapps\struts2tutorial\css" directory.
   Here is the code of main.css:

   ```css
   @CHARSET "UTF-8";
   
   body {
   font: 12px verdana, arial, helvetica, sans-serif;
   background-color:#FFFFFF;
   }
   
   table.wwFormTable {
   font: 12px verdana, arial, helvetica, sans-serif;
   border-width: 1px;
   border-color: #030;
   border-style: solid;
   color: #242;
   background-color: #ada;
   width: 30%;
   margin-left:35%;
   margin-right:35%;
   margin-top:15%;
   }
   
   table.wwFormTable th {
   ```
table.wwFormTable tr td {
  background-color: #dfd;
  margin: 5px;
  padding: 5px;
}

.tdLabel {
  /*
   * border-width: 1px;
   * border-color: #afa;
   * border-style: solid;
   */
  font-weight: bold;
  align: top;
}

.label {
}

.errorMessage {
  color: red;
  font-size: 0.8em;
}

#headerDiv {
  border-style: solid;
  border-width: 1px 1px 0px;
  border-color: black;
  padding: 5px;
  background-color: #7a7;
  /* height: 22px; */
  /* margin-bottom: 12px; */
  height: 1.8em;
}

#buttonBar {
  border-width: 0px 1px 1px;
  border-style: solid;
  border-color: black;
  color: white;
  margin-bottom: 12px;
  background-color: #7a7;
  height: 1.6em;
  padding: 5px;
Compiling the application

To compile the application go to "\webapps\struts2tutorial\WEB-INF\src" directory and type ant command. The ant tool will compile the application for you.

Adding the link into index.html

Finally we add the link in the index.html to access the login form.

```html
<ul>
<li><a href="roseindia/LoginAnnotation.action">Action Annotation Example</a></li>
</ul>
```

Output:

If you click Login button without filling the fields, you will get the output pages as:
If you fill only the "Login name" field and click Login button without filling the next fields, you will get the output pages as:

If you fill the wrong information and click the Login button, you will get the output pages as:

* Invalid user name or password! Please try again!

If you fill the correct information and click the Login button, you will get the output pages as:
In this section, we are going to develop a login/logout application with session. This application checks the user authentication. Whenever you run, it takes a user id and a password (Both the user id and password is "admin") it displays the welcome page, when both fields are correctly filled.

Create an action mapping in the struts.xml file. Here is the code to be added in the struts.xml:

```
<action name="login" class="net.roseindia.loginAction">
  <result name="success" type="dispatcher">/pages/uiTags/Success.jsp</result>
  <result name="error" type="redirect">/pages/uiTags/Login.jsp</result>
</action>

<action name="logout" class="net.roseindia.logoutAction">
  <result name="success" type="redirect">/pages/uiTags/checkLogin.jsp</result>
</action>
```

Develop an action class that handles the login request. The Struts 2 framework provides a base `ActionSupport` class that implements commonly used framework interfaces. In our action class (`loginAction.java`) we have extended `ActionSupport` class.
Here is the code of "loginAction" action class:

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

public class loginAction extends ActionSupport {
    private String userId;
    private String password;

    public String execute() throws Exception{
        if ("admin".equals(userId) && "admin".equals(password)) {
            Map session = ActionContext.getContext().getSession();
            session.put("logged-in","true");
            return SUCCESS;
        }
        else{
            return ERROR;
        }
    }

    public String logout() throws Exception {
        Map session = ActionContext.getContext().getSession();
        session.remove("logged-in");
        return SUCCESS;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getUserId() {
        return userId;
    }

    public void setUserId(String userId) {
        this.userId = userId;
    }
}
```

Download this code.

Again, develop an action class to handle the logout operation. An action class (logoutAction) we have extended ActionSupport class.

Here is the code of logoutAction action class:

```java
package net.roseindia;
```
import javax.servlet.http.HttpSession;
import com.opensymphony.xwork2.ActionSupport;
import com.opensymphony.xwork2.ActionContext;
import java.util.*;

public class logoutAction extends ActionSupport {
    public String execute() throws Exception {
        Map session = ActionContext.getContext().getSession();
        session.remove("logged-in");
        return SUCCESS;
    }
}

**Download this code.**

**Develop Login Form:** The GUI of the application consists of login form (Login.jsp). The "Login.jsp" displays the login page to the user.

Here is the code of Login.jsp file:

```html
<%@ taglib prefix="s" uri="/struts-tags" %>
<%@ page language="java" contentType="text/html"%>

<html>
  <head>
    <title>Insert Data here!</title>
    <link href="<s:url value="/css/main.css"/>" rel="stylesheet" type="text/css"/>
  </head>
  <body>
    <s:form action="/roseindia/login.action" method="POST">
      <s:textfield name="userId" label="Login Id"/><br>
      <s:password name="password" label="Password"/><br>
      <s:submit value="Login" align="center"/>
    </s:form>
  </body>
</html>
```

The "Success.jsp" page displays the Login Success message (Welcome, you have logged-in.) and session (Session Time: Wed Aug 01 11:26:38 GMT+05:30 2007 and Logout ) when user gets successful authentication. If you click the "Logout" then again login page is displayed on the screen.

Here is the code of Success.jsp file:

```html
<%@ taglib prefix="s" uri="/struts-tags" %>
<%@ page language="java" contentType="text/html" import="java.util.*"%>
<jsp:include page="/struts2tags/pages/uiTags/loginCheck.jsp" />

<html>
  <head>
    <title>Welcome, you have logined!</title>
  </head>
</html>
```
Welcome, you have logined. <br />
Session Time: <b><%= new Date(session.getLastAccessedTime()) %></b><br />
<a href="<%= request.getContextPath() %>/roseindia/logout.action" >Logout</a>
<br />
<br />
This page logs out the valid user.

checkLogin.jsp

<%@ taglib prefix="s" uri="/struts-tags" %>
<%@ page language="java" contentType="text/html" import="java.util.*" %>
<html>
    <head>
        <title>Check validate!</title>
    </head>
    <body>
        <s:if test="#session.login != 'admin'">
            <jsp:forward page="/pages/uiTags/Login.jsp" />
        </s:if>
    </body>
</html>

Output:

Run this application by getting the login page:

Enter the wrong user id and password in the login page
Enter the correct user id and password in the login page:
You get the following output:

Welcome, you have logined.
**Session Time:** Wed Aug 01 10:24:24 GMT+05:30 2007

Logout

After clicking the "Logout". You get the following output:
In this section, You will learn to connect the MySQL database with the struts 2 application.

Follow the following steps to connect with MySQL database:

**Step 1:** Create the struts.xml file and add the following xml snippet in the struts.xml file.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts
    Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
```
<constan
name="struts.enable.DynamicMethodInvocation"
value="false"/>

<constant name="struts.devMode" value="true"/>
<include file="struts-default.xml"/>

<!-- Add packages here -->

<package name="roseindia" namespace="/roseindia"
extends="struts-default">

<!-- inserting data into data base through JDBC -->

<action name="insert">
  <result>/pages/insertData.jsp</result>
</action>

<action name="insertData" class="net.roseindia.insert">
  <result name="error">/pages/insertData.jsp</result>
  <result>/pages/insertSuccess.jsp</result>
</action>

</package>
</struts>

**Step 2 : Create an input jsp form.**

```jsp
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
<head>
<title>Struts 2 Insert Data Application!</title>

<link href="<s:url value="/css/main.css"/>" rel="stylesheet" type="text/css"/>

</head>
<body>

<s:form action="insertData" method="POST" validate="true">

<tr>
<td colspan="2">Please enter</td>
</tr>
</s:form>
</body>
</html>
```
Step 3: Create an Action class.

First, Establish a connection with the MySQL Database with the help of MySQL driver ("org.gjt.mm.mysql.Driver").

Now, Make an account in the MySQL database to get connected with the database.

After establishing a connection, you can retrieve, insert and update data to the MySQL database table.

The following action class establishes a connection with MySQL database with the help of appropriate type of methods and API interfaces. If connection is established then the entered data is added to the MySQL database table otherwise it displays an error message.

insert.java

```java
package net.roseindia;
import com.opensymphony.xwork2.ActionSupport;
import java.util.Date;
import java.sql.*;

/**
 * <p> Validate a user login. </p>
 */
public class insert extends ActionSupport {

    public String execute() throws Exception {
        String url = "jdbc:mysql://localhost:3306/";
        String dbName = "taskproject";
        String driverName = "org.gjt.mm.mysql.Driver";
        String userName = "root";
        String password = "root";
        Connection con=null;
        Statement stmt=null;
        try{
            Class.forName(driverName).newInstance();
            con=DriverManager.getConnection(url+dbName, userName, password);
        } catch (Exception e) {
            insert.addActionError("Error in connecting MySQL database. "+e.getMessage());
        } finally {
            if (con != null) con.close();
        }
        // insert query code here
    }
}"
stmt=con.createStatement();
}
catch(Exception e){
    System.out.println(e.getMessage());
}
String uname=getUsername();
String pws=getPassword();
stmt = con.createStatement();
    int val = stmt.executeUpdate("INSERT employee VALUES ('"+uname+"','"+pws+"')");

if(val == 0){
    return ERROR;
}
else{
    return SUCCESS;
}
//-- **** Username property ----
/**
 * <p>Field to store User username.</p>
 * <p/>
 */
private String username = null;

/**
 * <p>Provide User username.</p>
 * @return Returns the User username.
 */
public String getUsername() { 
    return username;
}

/**
 * <p>Store new User username</p>
 * @param value The username to set.
 */
public void setUsername(String value) {
    username = value;
}
//-- **** Username property ----
/**
 * <p>Field to store User password.</p>
 * <p/>
 */
private String password = null;

/**
 * <p>Provide User password.</p>
 *
Description of the code:

**Connection:**
This is an interface in `java.sql` package that specifies establishing connection with the specific database like: MySQL, Ms-Access, Oracle etc and java files. The SQL statements are executed within the context of the Connection interface.

**Class.forName(String driver):**
This method is static. It attempts to load the class dynamically and returns class instance and takes string type value (driver) when it matches with the class with given string.

**DriverManager:**
It is a class of `java.sql` package that controls a set of JDBC drivers. Each driver has to be registered with this class.

**getConnection(String url, String userName, String password):**
This method establishes a connection to specified database `url`. It takes three string types of arguments like:

- `url`: Database url to link with
- `userName`: User name of database
- `password`: Password of database

**con.close():**
This method is used for disconnecting the connection. It frees all the resources occupied by the database.

**Step 4:** Create the validator
The validation.xml format is either <ActionClassName>-validation.xml or <ActionClassName>-<ActionAliasName>-validation.xml.

insert-validation.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator 1.0.2//EN" "http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">
<validators>
  <field name="username">
    <field-validator type="requiredstring">
      <param name="trim">true</param>
      <message>User name is required</message>
    </field-validator>
  </field>
  <field name="password">
    <field-validator type="requiredstring">
      <param name="trim">true</param>
      <message>Password is required</message>
    </field-validator>
  </field>
</validators>
```

When entered the correct data in the text field the user gets the insertSuccess.jsp page displaying the entered data.

insertSuccess.jsp

```html
<html>
<head>
<title>Inserted Data List</title>
</head>
<body>
<b>Inserted Data: </b><br/>
User name = <%=request.getParameter("username") %><br/>
Password = <%=request.getParameter("password") %>
</body>
</html>
```

Output:

When this application executes the user gets the following:
Without filling fields and click "Save" button, you will get the output page as:

If you fill only the "Password" field and click "Save" button without filling the next fields, you will get the output page as:

If you fill only the "User Name" field and click "Save" button without filling the next fields, you will get the output page as:
If you fill both field:

<table>
<thead>
<tr>
<th>Please enter</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>User Name:</strong></td>
<td>Vinod</td>
</tr>
<tr>
<td><strong>Password:</strong></td>
<td>********</td>
</tr>
</tbody>
</table>

*Password is required*

Then you get:

**Inserted Data:** User name = Vinod, Password = Vinod!

**Rich Editor Example**

In this section, you will learn how to create pagination in struts 2. For creating pagination in your application follows the certain steps:

**Download this code**

Step 1: Create **index.jsp** page

Here is the code to be added in the **index.jsp**:

```html
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
<title>RoseIndia.Net Struts 2 Tutorial</title>
</head>
```
Step 2: Create an action mapping in the struts.xml file.

Here is the code to be added in the struts.xml:
Step 3: Create a JSP page that contains pagination code:

Here is the code to be added in the pagination.jsp:

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

<html>
<head>
<title>Pagination Example</title>
<s:head theme="ajax" />
</head>

<body>
<s:form>
<s:textarea name="text" label="Paste your text here" theme="ajax"
            cssStyle="background-color: #FCFCFC; border: 1px solid #A0A0A0; min-height:500px; max-height:inherit; "/>
</s:form>
```
Output:

Paste your text here:

Validate TextArea

In this section, you will learn how to validate your text area in struts 2. A text area contains 1 to 250 characters. It can't support "null" value.

For validating your text area in your application follows the certain steps:

Step 1: Create index.jsp page
Here is the code to be added in the index.jsp:

```html
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
<title>RoseIndia.Net Struts 2 Tutorial</title>
</head>
<body>
<div align="center">
<table border="0" cellpadding="0" cellspacing="0" width="400">
<tr>
<td><font color="#000080" size="5"><b>RoseIndia.net Struts 2 Tutorials</b></font></td>
</tr>
<tr>
<td><font color="#000080">Select the following links to test the examples</font></td>
</tr>
<tr>
<td>
<ul>
<li><a href="roseindia/characterLimit.action">Characters and limiting Example</a></li>
</ul>
</td>
</tr>
<tr>
<td><font color="#000080">Visit <a>Visit</a></font></td>
</tr>
</table>
</div>
</body>
</html>
```
Step 2: Create an action mapping in the *struts.xml* file.

Here is the code to be added in the *struts.xml*:

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN" "http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>
  <!-- Rose India Struts 2 Tutorials -->
  <constant name="struts.enable.DynamicMethodInvocation" value="false" />
  <constant name="struts.devMode" value="true" />

  <package name="roseindia" namespace="/roseindia" extends="struts-default">
    <action name="characterLimit">
      <result>/pages/chatQuestion/limitedCharacter.jsp</result>
    </action>

    <action name="characterLimit1" class="net.roseindia.limitedCharacterAction">
      <result name="error">/pages/chatQuestion/limitedCharacter.jsp</result>
      <result name="success">/pages/chatQuestion/limitedCharacterSuccess.jsp</result>
    </action>
  </package>

  <!-- Add packages here -->

</struts>
```
Step 3: Create a JSP page that contains text area and submit button:

Here is the code to be added in the limitedCharacter.jsp:

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

<html>
<head>
<title>Limited Characters in Text Area</title>
<s:head theme="ajax"/>
</head>

<body>
<s:form action="characterLimit1" method="POST" validate="true">
<s:textarea name="summary1" label="Brief Summary" rows="6" cols="40"/>
<s:submit value="Save" align="center"/>
</s:form>
</body>
</html>
```

Step 4: Create an action class:

Here is the code to be added in the limitedCharacterAction.java:

```java
package net.roseindia;

import com.opensymphony.xwork2.ActionContext;
import com.opensymphony.xwork2.ActionSupport;
import java.io.PrintStream;
import java.util.List;
import java.util.*;

public class limitedCharacterAction extends ActionSupport {
    private String summary1;

    public String getSummary1() {
        return summary1;
    }

    public void setSummary1(String summary1) {
        this.summary1 = summary1;
    }
}
```
public String execute() throws Exception {
    String summaryText = getSummary1();
    long countSummaryText = summaryTextCount(summaryText);
    System.out.println("countSummaryText:" + countSummaryText);
    if (countSummaryText > 0 && countSummaryText < 250) {
        return SUCCESS;
    } else {
        if (getSummary1().equals(""))
            addFieldError("summary1","Brief Summary is required.");
        if (countSummaryText > 250)
            addFieldError("summary1","Brief Summary must be 1 to 250 Characters");
        return ERROR;
    }
}

private static long summaryTextCount(String str) {
    return str.length();
}

Step 5: Create a JSP page that contains the inputted text in the textarea:

Here is the code to be added in limitedCharacterSuccess.jsp:

```jsp<%@ taglib prefix="s" uri="/struts-tags"%>
<html>
<head>
<title>Limited Characters in Text Area</title>
<link href="<s:url value="/css/main.css"/>" rel="stylesheet" type="text/css"/>
</head>
<body>
<b>Brief Summary: </b><s:property value="summary1" /><br/>
</body>
</html>```
Output:

Brief Summary:

When you click the "Save" command button with out any data then you get:

**Brief Summary is required.**

If you enter more than 250 characters then you get:

**Brief Summary must be 1 to 250 Characters**

Brief Summary: Asad
Hello. How do I represent the following in a message resources file? Representing a string literal is easy but the following string contains a link. Can this even be done? I have used variables with the

If you enter your text between 1 to 250 characters then you get:

**Brief Summary:** Asad Hello. How do I represent the following in a message resources file? Representing a string literal is easy but the following string

Struts 2 Tags
In this section we will introduce you with the tags provided along with Struts 2 framework. It is necessary to understand all the tags provided along with Struts 2 framework. In this page we will have listed all the Struts 2 Tags and in subsequent sections we will provide you the examples of the these tags in detail.

The Struts 2 Tags can be divided into two types:

- **Struts Generic Tags**
  The struts generic tags are used to control the execution flow when pages are rendered. Another use of struts generic tags are data extraction. Further Generic Tags are classified into Control Tags and Data Tags.

  **Control Tags:** The Control Tags are used for flow control, such as if, else and iterate.

  **Here are the list of Control Tags:**
  
  * if
  * elseif
  * else
  * append
  * generator
  * iterator
  * merge
  * sort
  * subset

  **Data Tags:** The Data Tags are used for data manipulation or creation, such as bean, push, and i18n.

  **Here are the list of Data Tags:**
  
  * a
  * action
  * bean
  * date
  * debug
  * i18n
  * include
  * param
  * push
  * set
Struts UI tags
Struts UI Tags are mainly designed to use the data from your action/value stack or from Data Tags. These tags are used to display the data on the HTML page. The UI tags are driven by templates and themes.
Struts UI Tags are of two types Form Tags and Non-Form tags.
Form Tags are as follows:
- autocompleter
- checkbox
- checkboxlist
- combobox
- datetimepicker
- doubleselect
- head
- file
- form
- hidden
- label
- optiontransferselect
- optgroup
- password
- radio
- reset
- select
- submit
- textarea
- textfield
- token
- updownselect

Non-Form UI Tags are:
- actionerror
- actionmessage
- component
- div
- fielderror
- table
- tabbedPanel
- tree
- treenode

The main difference between Struts Generic Tags and Struts UI Tags are:
The generic tags simply output some content directly from the tag while the UI tags uses templates and often group the output together with theme

Control Tags-If / Else If / Else

In this section we are going to discuss the various control tags (The Control Tags are used for flow control such as if, else and iterate.)

'If' tag could be used by itself or with 'Else If' Tag and/or single/multiple 'Else' Tag.

Create a JSP page IfControlTag.jsp.

Set a property 'technologyName' with a value 'Java' as
<s:set name="technologyName" value="%{'Java'}"/>

Among if, elseif and else tags only one tag evaluates at a time. Evaluation is based upon the conditions being processed. Evaluated conditions must be of Boolean type. This is illustrated in the following Jsp page.

[Note:
If the condition in <s:if > tag evaluates to 'true' then only this tag is evaluated and others are discarded. As illustrated in the example.

If the condition in <s:if > tag evaluates to 'false' and <s:elseif > tag evaluates to 'true' then the body of the <s:elseif > tag is processed.

If the condition in <s:if > tag and <s:elseif > tags evaluates to 'false' then only the <s:else > tag is processed. ]

IfControlTag.jsp

<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
<head>
<title>Struts 2 Control Tag Example</title>
</head>
<body>
<s:set name="technologyName" value="%{'Java'}"/>

<s:if test="%{#technologyName=='Java'}">
<div><s:property value="%{#technologyName}" /></div>
</s:if>

<s:elseif test="%{#technologyName=='Java1'}">
<div><s:property value="%{#technologyName}" /></div>
</s:elseif>
</body></html>
**struts.xml:** Add the following xml snippet in the struts.xml file.

```xml
<action name="dolf">
  <result>/pages/genericTags/IfControlTag.jsp</result>
</action>
```

**index.jsp:** Add the following jsp snippet in the index.jsp file.

```jsp
<ul>
  <li><a href="roseindia/dolf.action">IF Control Tag Example</a></li>
</ul>
```

In the IfControlTag.jsp only `<s:if>` tag evaluates to true

```jsp
<s:if test="%{#technologyName=='Java'}">
  <div><s:property value="%{#technologyName}" /></div>
</s:if>
```

So we get the output equal to Java

### Append Tag (Control Tags) Example

In this section, we are going to describe the append tag. The append tag is a generic tag that is used to merge multiple iterators into one iterator.  
**Append Iterator Tag** is used to append iterators to form an appended iterator through which the entries goes from one iterator to another after each respective iterator is exhausted of entries.

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

**struts.xml** :

```xml
<action name="AppendTag" class="net.roseindia.AppendTag">
```

```xml
</s:else>
  <div>Technology Value is not Java</div>
</s:else>
</body>
</html>
```
Create two lists in the action class and populate them with various items as shown in the "AppendTag" class.

**AppendTag.java**

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

public class AppendTag extends ActionSupport{

    private List myList;
    private List myList1;

    public String execute() throws Exception{
        myList = new ArrayList();
        myList.add("www.Roseindia.net");
        myList.add("Deepak Kumar");
        myList.add("Sushil Kumar");
        myList.add("Vinod Kumar");
        myList.add("Amit Kumar");

        myList1 = new ArrayList();
        myList1.add("www.javajazzup.com");
        myList1.add("Himanshu Raj");
        myList1.add("Mr. khan");
        myList1.add("John");
        myList1.add("Ravi Ranjan");
        return SUCCESS;
    }

    public List getMyList(){
        return myList;
    }

    public List getMyList1(){
        return myList1;
    }
}
```

Now create a jsp page using `<s:append>` and `<s:iterator>` tags as shown in the `AppendTag.jsp` page. The append tag is used to merge multiple iterators into one iterator. The "id" parameter keeps the resultant appended iterator stored under the stack's context and the "value" parameter is used to get the values contained within the resultant iterator.

**AppendTag.jsp**
Output of the Append Tag Example:

Append Tag Example!

www.Roseindia.net
Deepak Kumar
Sushil Kumar
Vimod Kumar
Amit Kumar
www.javajazzup.com
Himanshu Raj
Mr. khan
John
Ravi Ranjan
Generator Tag (Control Tags) Example

In this section, we are going to describe the generator tag. The generator tag is a generic tag that is used to generate iterators based on different attributes passed. Here we will not pass any attribute.

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

```xml
<action name="GeneratorTag"
      class="net.roseindia.GeneratorTag">
  <result>/pages/genericTags/GeneratorTag.jsp</result>
</action>
```

Create an action class as shown:

```java
package net.roseindia;
import com.opensymphony.xwork2.ActionSupport;
import org.apache.struts2.util.IteratorGenerator.Converter;

public class GeneratorTag extends ActionSupport {
    public String execute() throws Exception{
        return SUCCESS;
    }
}
```

Create a jsp page where the generator tag `<s:generator>` generates a simple iterator based on the `val` attribute supplied and `<s:iterator>` tag prints it out using the `<s:property />` tag. The `separator` attribute is used to separate the `val` into entries of the iterator.

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

<html>
<head>
    <title>Generator Tag Example!</title>
</head>
<body>
    <h1><span style="background-color: #FFFFcc">Generator Tag Example!</span></h1>
```
Generates a Simple Iterator

www.Roseindia.net
Deepak Kumar
Sushil Kumar
Vinod Kumar
Amit Kumar

generator tag using the count attribute.

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

```xml
<action name="GeneratorTagCountAttribute"
class="net.roseindia.GeneratorTag">
<result>/pages/genericTags/GeneratorTagCountAttribute.jsp</result>
</action>
```

Create an action class as shown:

GeneratorTag.java

```java
package net.roseindia;
import com.opensymphony.xwork2.ActionSupport;
import org.apache.struts2.util.IteratorGenerator.Converter;

public class GeneratorTag extends ActionSupport {
    public String execute() throws Exception {
        return SUCCESS;
    }
}
```

Create a jsp page where the generator tag `<s:generator>` generates an iterator with "count" attribute and `<s:iterator>` tag prints it out using the `<s:property>` tag. The separator attribute separates the `val` into entries of the iterator.

This generates an iterator, but only 5 entries will be available in the iterator generated, namely 'www.Roseindia.net, Deepak Kumar, Sushil Kumar, Vinod Kumar, Amit Kumar' respectively because count attribute is set to 5.

GeneratorTagCountAttribute.jsp

```jsp
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
    <head>
        <title>Generator Tag Example! </title>
    </head>
    <body>
        <h1><span style="background-color: #FFFFcc">Generator Tag Example!</span></h1>
        <h3><font color="#0000FF"> Generates an Iterator With Count Artribute</font></h3>
        <s:generator val="%{'www.Roseindia.net,Deepak Kumar,Sushil Kumar, Vinod Kumar, Amit Kumar,'}" count="5" separator=";">
            <s:iterator>
                <s:property /><br/>
            </s:iterator>
        </s:generator>
    </body>
</html>
```
Output of the Generator Tag Example:

**Generator Tag Example!**

Generates an Iterator With Count Attribute

www.Roseindia.net  
Deepak Kumar  
Sushil Kumar  
Vinod Kumar  
Amit Kumar

Generator Tag (Control Tags) Using an Iterator with Id Attributes

In this section, we are going to describe the generator tag using the id attributes.

Add the following code snippet into the `struts.xml` file.
Create an action class as shown:

**GeneratorTag.java**

```java
package net.roseindia;
import com.opensymphony.xwork2.ActionSupport;
import org.apache.struts2.util.IteratorGenerator.Converter;

class GeneratorTag extends ActionSupport {

    public String execute() throws Exception{
        return SUCCESS;
    }
}
```

Create a jsp page where the generator tag `<s:generator>` generates an iterator with "id" attribute and `<s:iterator>` tag prints out using the `<s:property />` tag. The `separater` attribute separates the value into entries of the iterator.

The scriplet as shown below generates an iterator and put it in the `PageContext` under the key as specified by the id attribute.

```jsp
<% @taglib prefix="s" uri="/struts-tags" %>
<%@page language="java" import="java.util.*"%>

<html>
<head>
    <title>Generator Tag Example!</title>
</head>
<body>
    <h1><span style="background-color: #FFFFcc">Generator Tag Example!</span></h1>
    <h3><font color="#0000FF">Generates an Iterator With Id Attribute</font></h3>
```
Output of the Generator Tag Example:

Generates an Iterator With Id Attribute

www.Roseindia.net
Deepak Kumar
Sushil Kumar
Vinod Kumar

Iterator Tag (Control Tags) Example

In this section, we are going to describe the Iterator tag. Iterator tag is used to iterate over a value. An iterable value can be either of: java.util.Collection, java.util.Iterator.
Add the following code snippet into the `struts.xml` file.

```xml
<action name="iteratorTag"
class="net.roseindia.iteratorTag">

<result>/pages/genericTags/iteratorTag.jsp</result>

</action>
```

Create an action class as shown:

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

public class iteratorTag extends ActionSupport{

    private List myList;

    public String execute() throws Exception{
        myList = new ArrayList();
        myList.add("Fruits");
        myList.add("Apple");
        myList.add("Mango");
        myList.add("Orange");
        myList.add("Pine Apple");
        return SUCCESS;
    }

    public List getMyList(){
        return myList;
    }
}
```

The following example retrieves the value of the `getMyList()` method of the current object on the value stack and uses it to iterate over. The `<s:property/>` tag prints out the current value of the iterator.

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

<html>
    <head>
        <title>Iterator Tag Example!</title>
    </head>
    <body>
```
Output of An Iterator Tag Example:

<table>
<thead>
<tr>
<th>Fruits</th>
<th>Apple</th>
<th>Mango</th>
<th>Orange</th>
<th>Pine Apple</th>
</tr>
</thead>
</table>

Merge Tag (Control Tags) Example

In this section, we are going to describe the merge tag. The merge tag is a generic tag that is used to merge iterators. The successive call to the merge iterator causes each merge iterator to have a chance to expose its element, subsequently next call allows the next iterator to expose its element. Once the last iterator is done exposing its element, the first iterator is allowed to do so again (unless it is exhausted of entries).

In the current example, 2 lists being merged, each list have 5 entries, the following will be the logic.

1. Display first element of the first list.
2. Display first element of the second list.
3. Display second element of the first list.
4. Display second element of the second list.
5. Display third element of the first list.
6. Display third element of the second list.....and so on.

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

**struts.xml**

```xml
<action name="mergeTag"
class="net.roseindia.mergeTag">
  <result>/pages/genericTags/mergeTag.jsp</result>
</action>
```

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

**mergeTag.java**

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

public class mergeTag extends ActionSupport {
    private List myList;
    private List myList1;

    public String execute() throws Exception {
        myList = new ArrayList();
        myList.add("www.Roseindia.net");
        myList.add("Deepak Kumar");
        myList.add("Sushil Kumar");
        myList.add("Vinod Kumar");
        myList.add("Amit Kumar");

        myList1 = new ArrayList();
        myList1.add("www.javajazzup.com");
        myList1.add("Himanshu Raj");
        myList1.add("Mr. Khan");
        myList1.add("John");
        myList1.add("Ravi Ranjan");
        return SUCCESS;
    }

    public List getMyList() {
        return myList;
    }

    public List getMyList1() {
        return myList1;
    }
}
```
Now create a jsp page using `<s:merge>` and `<s:param value>` tags as shown in the `mergeTag.jsp` page. The `merge` tag is used to merge iterators. The "id" parameter keeps the resultant iterator stored under in the stack's context and the "value" parameter in the `<s:iterator>` is used to get the values contained within the respective iterators.

mergeTag.jsp

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

<html>
<head>
    <title>Merge Tag Example!</title>
</head>
<body>
    <h1><span style="background-color: #FFFFcc">Merge Tag Example!</span></h1>
    <s:merge id="mergeId">
        <s:param value="%{myList}" />
        <s:param value="%{myList1}" />
    </s:merge>
    <s:iterator value="%{#mergeId}"
        <s:property /></s:iterator>
</body>
</html>
```

Output of merge Tag:
Subset Tag (Control Tags) Example

In this section, we are going to describe the subset tag. The subset tag is a generic tag that takes an iterator and outputs a subset of it. It delegates to org.apache.struts2.util.SubsetIteratorFilter internally to perform the subset functionality.

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

```xml
<action name="subsetTag" class="net.roseindia.subsetTag">
<result>/pages/genericTags/subsetTag.jsp</result>
</action>
```

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

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

public class subsetTag extends ActionSupport {
    private List myList;

    public String execute() throws Exception{
        myList = new ArrayList();
        myList.add(new Integer(50));
        myList.add(new Integer(20));
        myList.add(new Integer(100));
        myList.add(new Integer(85));
        myList.add(new Integer(500));
        return SUCCESS;
    }

    public List getMyList(){
        return myList;
    }
}
```

Now create a jsp page using `<s:subset>` and `<s:iterator>` tags as shown in the subsetTag.jsp page. The subset tag takes an iterator and outputs a subset of it.

subsetTag.jsp

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

<html>
<head>
    <title>Subset Tag Example!</title>
</head>
<body>
    <h1><span style="background-color: FFFcc">Subset Tag Example!</span></h1>
    <s:subset source="myList">
        <s:iterator>
            <s:property /><br>
        </s:iterator>
    </s:subset>
</body>
</html>
```

Output of Subset Tag:
Subset Tag Example!

50
20
100
85
500