

# ICON OF JAVA

# ADV. JAVA JSP

## 10. JSP Custom Actions



**Mr. Nagoor Babu** M.Tech

**[ICON of JAVA]**

**(Sun certified & Realtime Expert)**

**Ex. HCL Employee**

**Trained Lakhs of Students  
for last 14 years across INDIA**

**India's No.1 Software Training Institute**

# DURGASOFT

**www.durgasoft.com Ph: 9246212143 ,8096969696**

## JSP Custom Actions

In Jsp technology, by using Jsp directives we are able to define present Jsp page characteristics, we are unable to use Jsp directives to perform actions in the Jsp pages.

To perform actions if we use scripting elements then we have to provide Java code inside the Jsp pages.

The main theme of Jsp technology is not to allow Java code inside Jsp pages, to eliminate Java code from Jsp pages we have to eliminate scripting elements.

If we want to eliminate scripting elements from Jsp pages we have to use actions.

There are 2 types of actions in Jsp technology.

1. Standard Actions
2. Custom Actions

Standard Actions are predefined actions provided by Jsp technology, these standard actions are limited in number and having bounded functionalities so that standard actions are not sufficient to satisfy the complete client requirement.

In this context, there may be a requirement to provide Java code inside the Jsp pages so that to eliminate Java code completely from Jsp pages we have to use Custom Actions.

**[www.durgasoftonlinelearning.com](http://www.durgasoftonlinelearning.com)**



**Online Training  
Pre Recorded Video  
Classes Training  
Corporate Training**

**Ph: +91-8885252627, 7207212427  
+91-7207212428**

 **USA Ph : 4433326786**

**E-mail : [durgasoftonlinelearning@gmail.com](mailto:durgasoftonlinelearning@gmail.com)**

# DURGASOFT Means JAVA JAVA Means DURGASOFT



**India's No.1 JAVA Trainer**

More Than 1000 Students in a Single Class Room.

**Weapon of DURGASOFT**

**Mr. Nagoor Babu** M.Tech  
Trained Lakhs of Students for Last 13 years (Realtime Expert & Java Certified)

**Custom Actions** are Jsp Actions which could be prepared by the developers as per their application requirements.

In Jsp technology, standard actions will be represented in the form of a set of predefined tags are called as **Action Tags**.

Similarly all the custom actions will be represented in the form of a set of user defined tags are called as **Custom Tags**.

To prepare custom tags in Jsp pages we have to use the following syntax.

**Syntax:** <prefix\_Nmae: tag\_Name>

```
-----  
          ----- } Body  
-----  
</prefix_Name>
```





If we want to design custom tags in our Jsp applications then we have to use the following 3 elements.

1. Jsp page with taglib directive
2. TLD(Tag Library Descriptor) file
3. TagHandler class

Where TagHandler class is a normal Java class it is able to provide the basic functionality for the custom tags.

Where TLD file is a file, it will provide the mapping between custom tag names and respective TagHandler classes.

Where taglib directive in the Jsp page can be used to make available the tld files into the present Jsp page on the basis of custom tags prefix names.

## Internal Flow:

---

When container encounters a custom tag container will pick up the custom tag name and the respective prefix name then recognize a particular taglib directive on the basis of the prefix attribute value.

After recognizing taglib directive container will pick up uri attribute value i.e. the name and location of tld file then container will recognize the respective tld file.

After getting tld file container will identify the name and location of TagHandler class on the basis of custom tag name.

When container recognize the respective TagHandler class .class file then container will perform TagHandler class loading, instantiation and execute all the life cycle methods.

## 1. Taglib Directive:

In custom tags design, the main purpose of taglib directive is to make available the required tld file into the present Jsp page and to define prefix names to the custom tags.

**Syntax:** <%@taglib uri="--" prefix="--"%>

Where prefix attribute can be used to specify a prefix name to the custom tag, it will have page scope i.e. the specified prefix name is valid up to the present Jsp page.

Where uri attribute will take the name and location of the respective tld file.



**DURGASOFT Means JAVA**  
**JAVA Means DURGASOFT**

**Mr. Nagoor Babu** M.Tech.  
Trained Lakhs of Students for Last 13 years (Realtime Expert & Java Certified)

India's No.1 JAVA Trainer

More Than 1000 Students in a Single Class Room.

Weapon of DURGASOFT

## 2. TLD File:

The main purpose of TLD file is to provide the mapping between custom tag names and the respective TagHandler classes and it is able to manage the description of the custom tags attributes.

To provide the mapping between custom tag names and the respective TagHandler class we have to use the following tags.

```
<taglib>
<jsp-version>jsp version</jsp-version>
<tlib-version>tld file version</tlib-version>
<short-name>tld file short name</short-name>
<description>description about tld file</description>
<tag>
<name>custom tag name</name>
<tag-class>fully qualified name of TagHandler class</tag-class>
<body-content>jsp or empty</body-content>
```

```
<short-name>custom tag short name</short-name>
<description>description about custom tags</description>
</tag>
-----
</taglib>
```

**www.durgasoftonlinelearning.com**



**Online Training  
Pre Recorded Video  
Classes Training  
Corporate Training**

**Ph: +91-8885252627, 7207212427  
+91-7207212428**

 **USA Ph : 4433326786**

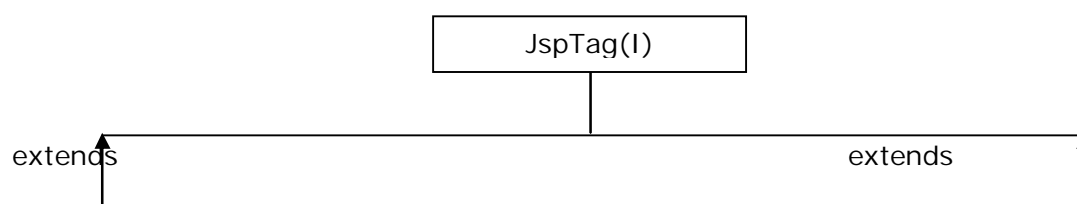
**E-mail : durgasoftonlinelearning@gmail.com**

### 3. TagHandler class:

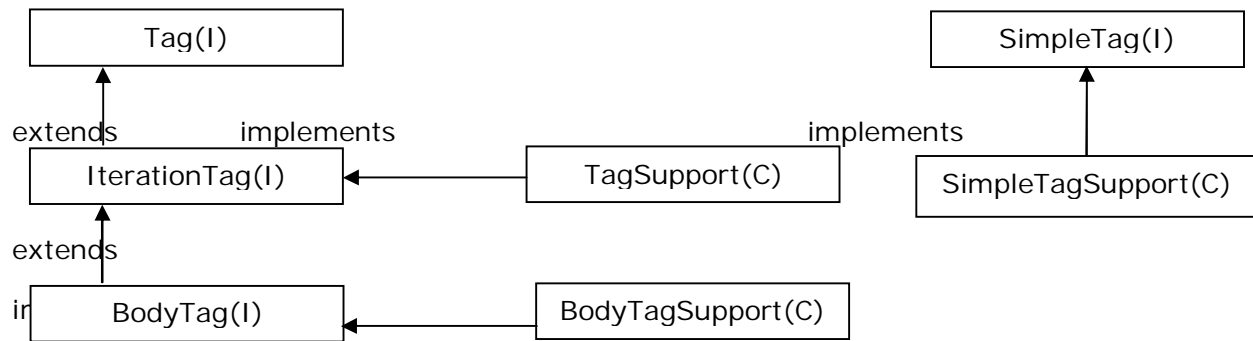
In custom tags preparation, the main purpose of TagHandler class is to define the basic functionality for the custom tags.

To design TagHandler classes in custom tags preparation Jsp API has provided some predefined library in the form of javax.servlet.jsp.tagext package (tagext-→tag extension).

javax.servlet.jsp.tagext package has provided the following library to design TagHandler classes.



**DURGASOFT, # 202,2<sup>nd</sup>Floor,HUDAMaitrivanam,Ameerpet, Hyderabad - 500038, ☎ 040 – 64 51 27 86,  
80 96 96 96 96, 9246212143 | www.durgasoft.com**



The above Tag library was divided into 2 types.

1. Classic Tag library
2. Simple Tag library

As per the tag library provided by Jsp technology there are 2 types of custom tags.

1. Classic tags
2. Simple Tags

LEARN FROM EXPERT & DIAMOND FACULTIES OF AMEERPET...

# JAVA MEANS DURGASOFT

INDIA'S NO. 1 SOFTWARE TRAINING INSTITUTE

AN ISO 9001:2008 CERTIFIED  
**DURGA**  
 SOFTWARE SOLUTIONS

**#202 2<sup>nd</sup> FLOOR**  
[www.durgasoft.com](http://www.durgasoft.com)

**040-64512786**  
**+91 9246212143**  
**+91 8096969696**

## 1. Classic Tags:



Classic Tags are the custom tags will be designed on the basis of Classic tag library provided by Jsp API.

As per the classic tag library provided by Jsp API there are 3 types of classic tags.

1. Simple Classic Tags
2. Iterator Tags
3. Body Tags



# DURGASOFT Means JAVA JAVA Means DURGASOFT



**India's No.1 JAVA Trainer**

More Than 1000 Students in a Single Class Room.

**Weapon of DURGASOFT**

**Mr. Nagoor Babu** M.Tech  
Trained Lakhs of Students for Last 13 years (Realtime Expert & Java Certified)

**FREE TRAINING VIDEOS**

**You Tube** **3000+ VIDEOS**

**[www.youtube.com/durgasoftware](http://www.youtube.com/durgasoftware)**

## 2. Simple Classic Tags:

Simple Classic Tags are the classic tags, which should not have body and attributes list.

To design simple classic tags the respective TagHandler class must implement Tag interface either directly or indirectly.

```
public interface Tag extends JspTag
{
    public static final int EVAL_BODY_INCLUDE;
    public static final int SKIP_BODY;
    public static final int EVAL_PAGE;
    public static final int SKIP_PAGE;
}
```



```

public void setPageContext(PageContext pageContext);
public void setParent(Tag t);
public Tag getParent();
public int doStartTag()throws JspException;
public int doEndTag()throws JspException;
public void release();
}
public class MyHandler implements Tag
{
    -----
    -----
}

```

Where the purpose of setPageContext(\_) method is to inject pageContext implicit object into the present TagHandler class.

Where the purpose of setParent(\_) method is to inject parent tags TagHandler class object into the present TagHandler class.

Where the purpose of getParent() method is to return the parent tags TagHandler class object from the TagHandler class.

Where the purpose of doStartTag() method is to perform a particular action when container encounters the start tag of the custom tag.

After the custom tags start tag evaluating the custom tag body is completely depending on the return value provided by doStartTag () method.

There are 2 possible return values from doStartTag() method.

1. EVAL\_BODY\_INCLUDE
2. SKIP\_BODY

If doStartTag() method returns EVAL\_BODY\_INCLUDE constant then container will evaluate the custom tag body.

If doStartTag() method returns SKIP\_BODY constant then container will skip the custom tag body and encounter end tag.

Where the purpose of doEndTag() method is to perform an action when container encounters end tag of the custom tag.

Evaluating the remaining the Jsp page after the custom tag or not is completely depending on the return value provided by doEndTag() method.

There are 2 possible return values from doEndTag() method.

1. EVAL\_PAGE
2. SKIP\_PAGE

If doEndTag() method returns EVAL\_PAGE constant then container will evaluate the remaining Jsp page.

If doEndTag() method returns SKIP\_PAGE constant then container will not evaluate the remaining Jsp page.

Where release() method can be used to perform TagHandler class deinstantiation.

**www.durgasoftonlinelearning.com**



**Online Training  
Pre Recorded Video  
Classes Training  
Corporate Training**

**Ph: +91-8885252627, 7207212427  
+91-7207212428**

 **USA Ph : 4433326786**

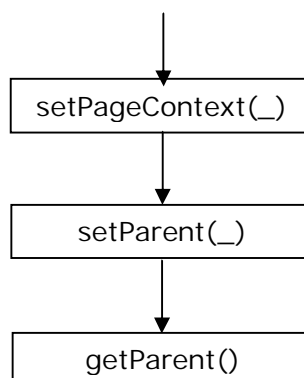
**E-mail : durgasoftonlinelearning@gmail.com**

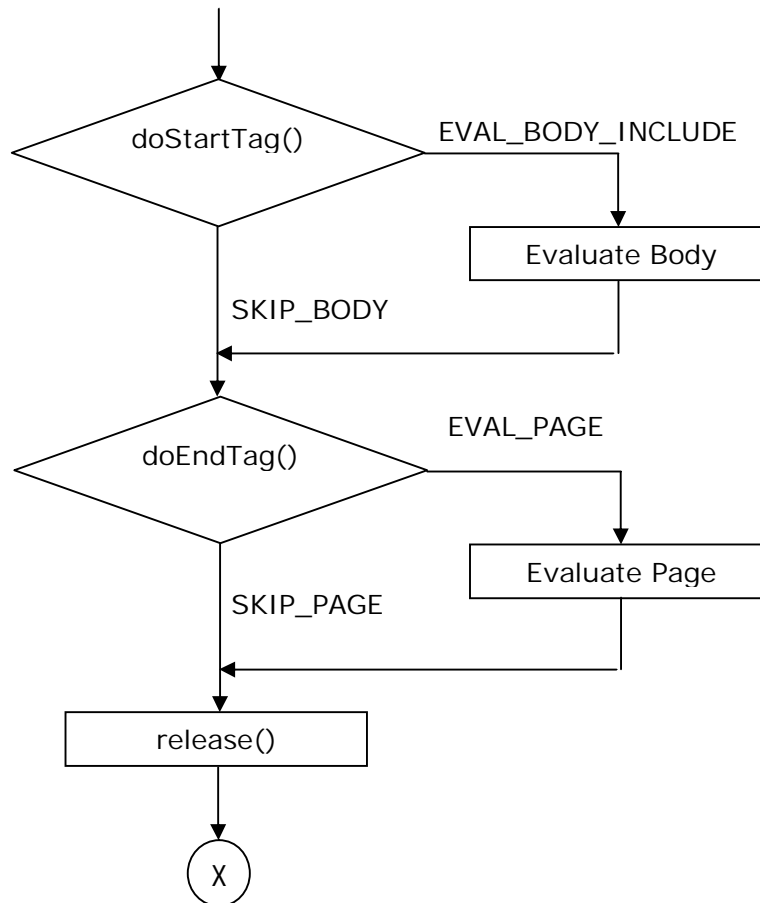
**FREE TRAINING VIDEOS**

**You Tube** **3000+ VIDEOS**

**www.youtube.com/durgasoftware**

## Life Cycle of Tag interface:





**Note:** In Tag interface life cycle, container will execute `getParent()` method when the present custom tag is child tag to a particular parent tag otherwise container will not execute `getParent()` method.

#### -----Application6-----

**custapp1:**

**hello.jsp:**

```
<%@taglib uri="/WEB-INF/hello.tld" prefix="mytags"%>
<mytags:hello/>
```

**hello.tld:**

```
<taglib>
  <jsp-version>2.1</jsp-version>
```



```

<tlib-version>1.0</tlib-version>
<tag>
    <name>hello</name>
    <tag-class>com.dss.HelloHandler</tag-class>
    <body-content>jsp</body-content>
</tag>
</taglib>

```

### **HelloHandler.java:**

```

package com.dss;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
public class HelloHandler implements Tag
{
    PageContext pageContext;
    public void setPageContext(PageContext pageContext)
    {
        this.pageContext=pageContext;
        System.out.println("setPageContext()");
    }
    public void setParent(Tag t)
    {
        System.out.println("setParent()");
    }
    public Tag getParent()
    {
        System.out.println("getParent()");
        return null;
    }
    public int doStartTag() throws JspException
    {
        try
        {
            System.out.println("doStartTag()");
            JspWriter out=pageContext.getOut();
            out.println("<h1>Hello..... First Custom Tag Application</h1>");
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
        return SKIP_BODY;
    }
    public int doEndTag()throws JspException
    {
        System.out.println("doEndTag()");
        return SKIP_PAGE;
    }
    public void release()
    {}
}

```

**Note:** To compile above code we need to set the classpath environment variable to the location of jsp-api.jar file.

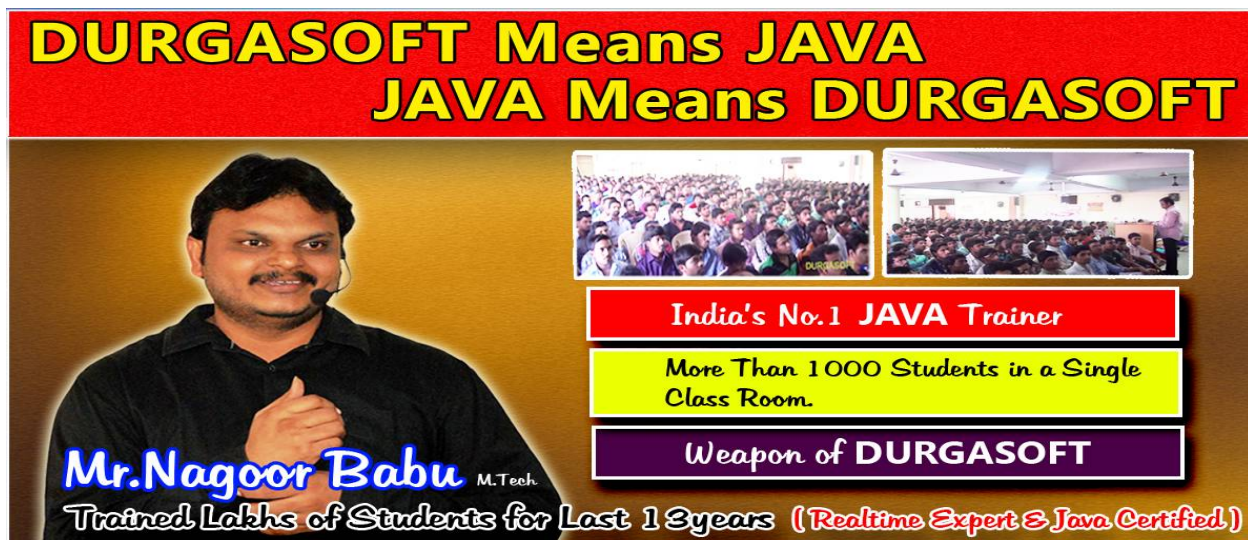
## Observations:

---

**Case 1:** In the above application, if we provide <body-content> type is empty in the tld file and if we provide body to the custom tag then container will raise an Exception like

org.apache.jasper.JasperException: /hello.jsp(2,0) According to TLD, tag mytags:hello must be empty, but is not.

**Case 2:** If we provide <body-content> value as jsp in tld file, if we provide body to custom tag in jsp page and if we return SKIP\_BODY constant in the respective TagHandler class then container won't raise any Exception but container won't evaluate custom tag body.



**DURGASOFT Means JAVA**  
**JAVA Means DURGASOFT**

**Mr. Nagoor Babu** M.Tech.  
Trained Lakhs of Students for Last 13 years (Realtime Expert & Java Certified)

India's No.1 JAVA Trainer

More Than 1000 Students in a Single Class Room.

Weapon of DURGASOFT

## Attributes in Custom Tgas:

---

If we want to provide attributes in custom tags then we have to perform the following steps.

**Step 1:** Define attribute in the custom tag.

**Ex:** <mytags:hello name="Durga"/>

**Step 2:** Provide attributes description in the respective tld file.

To provide attributes description in tld file we have to use the following tags in tld file.

<taglib>  
-----

```

<tag>
    -----
<attribute>
<name>attribute_name</name>
<required>true/false</required>
<rtexprvalue>true/false</rtexprvalue>
</attribute>
</tag>
</taglib>

```

Where <attribute> tag can be used to represent a single attribute in the tld file.  
Where <name> tag will take attribute name.

Where <required> tag is a boolean tag, it can be used to specify whether the attribute is mandatory or optional.

Where <rtexprvalue> tag can be used to specify whether the attribute accept runtime values or not.

**Step 3:** Declare a property and setter method in TagHandler class with the same name of the attribute defined in custom tag.

```

public class MyHandler implements Tag {
private String name;
public void setName(String name) {
    this.name=name;
}
    -----
}

```

LEARN FROM EXPERT & DIAMOND FACULTIES OF AMEERPET...

# JAVA MEANS DURGASOFT

INDIA'S NO. 1 SOFTWARE TRAINING INSTITUTE

AN ISO 9001:2008 CERTIFIED  
**DURGA**  
 SOFTWARE SOLUTIONS

**#202 2<sup>nd</sup> FLOOR**  
[www.durgasoft.com](http://www.durgasoft.com)

040-64512786  
 +91 9246212143  
 +91 8096969696

## 2. Iterator Tags:

---

Iterator tags are the custom tags, it will allow to evaluate custom tag body repeatedly.

If we want to prepare iterator tags the respective TagHandler class must implement javax.servlet.jsp.tagext.IterationTag interface.

```

public interface IterationTag extends Tag {
public static final int EVAL_BODY_INCLUDE;

public static final int SKIP_BODY;

public static final int EVAL_PAGE;

```



```

public static final int SKIP_PAGE;

public static final int EVAL_BODY_AGAIN;

public void setPageContext(PageContext pageContext);

public void setParent(Tag t);

public Tag getParent();

public int doStartTag()throws JspException;

public int doAfterBody()throws JspException;

public int doEndTag()throws JspException;

public void release();
}

public class MyHandler implements IterationTag
{ ----- }

```

In general there are 2 possible return values from doStartTag() method.

1. EVAL\_BODY\_INCLUDE
2. SKIP\_BODY

If we return SKIP\_BODY constant from doStartTag() method then container will skip the custom tag body.

If we return EVAL\_BODY\_INCLUDE constant from doStartTag() method then container will execute the custom tag body.

**Note:**In case of iterator tags, we must return EVAL\_BODY\_INCLUDE from doStartTag() method.

After evaluating the custom tag body in case of iterator tags, container will access doAfterBody() method.

In the above context, evaluating the custom tag body again or not is completely depending on the return value which we are going to return from doAfterBody() method.

1. EVAL\_BODY\_AGAIN
2. SKIP\_BODY

If we return EVAL\_BODY\_AGAIN constant from doAfterBody() method then container will execute the custom tag body again.

If we return SKIP\_BODY constant from doAfterBody() method then container will skip out custom tag body evaluation and encounter end tag of the custom tag.

# DURGASOFT Means JAVA

# JAVA Means DURGASOFT



**India's No.1 JAVA Trainer**

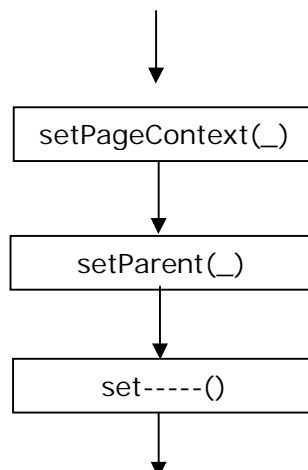
More Than 1000 Students in a Single Class Room.

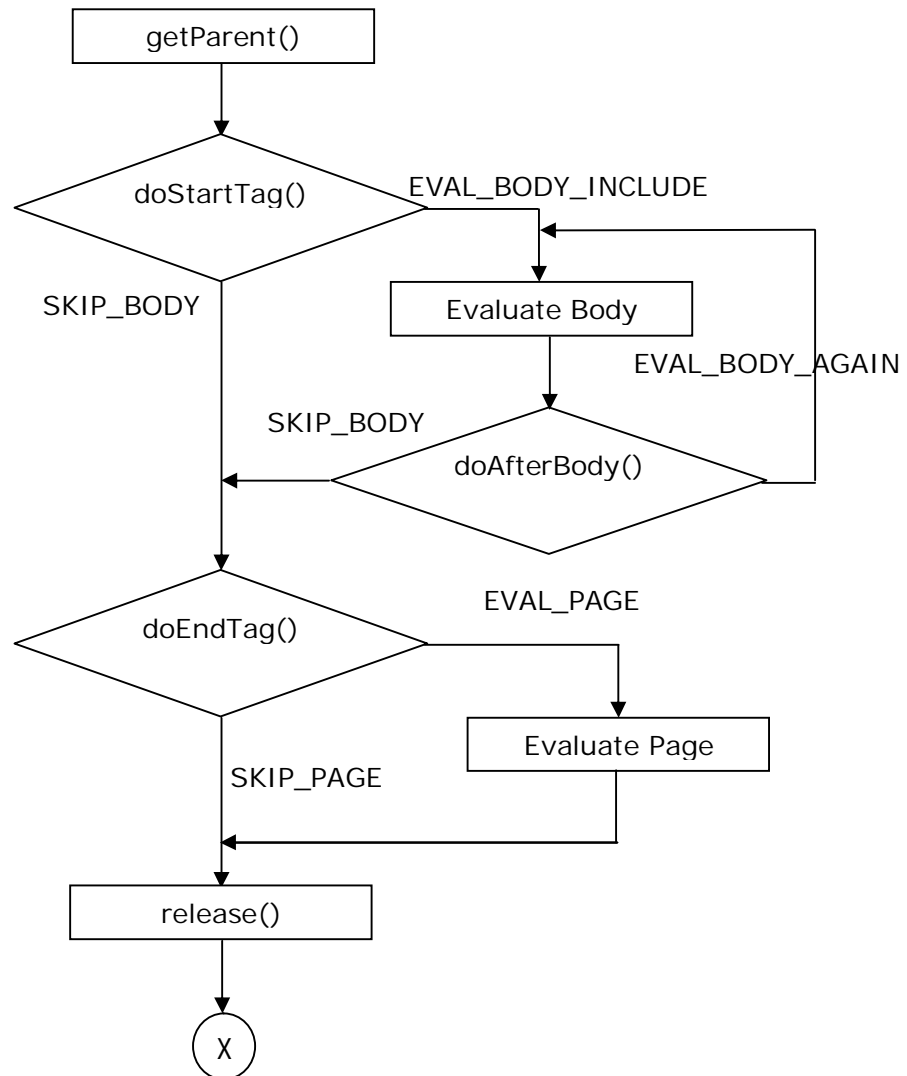
**Weapon of DURGASOFT**

**Mr. Nagoor Babu** M.Tech

Trained Lakhs of Students for Last 13years (Realtime Expert & Java Certified)

Life Cycle of IterationTag interface:





If we want to design custom tags by using above approach then the respective TagHandler class must implement Tag interface and IterationTag interface i.e. we must

provide the implementation for all the methods which are declared in Tag and IterationTag interfaces in our TagHandler class.

This approach will increase burden to the developers and unnecessary methods in TagHandler classes.

To overcome the above problem Jsp API has provided an alternative in the form of TagSupport class.

TagSupport is a concrete class, which was implemented Tag and IterationTag interfaces with the default implementation.

If we want to prepare custom tags with the TagSupport class then we have to take an user defined class, which must be a subclass to TagSupport class.

public interface TagSupport implements IterationTag {



```

public static final int EVAL_BODY_INCLUDE;
public static final int SKIP_BODY;
public static final int EVAL_PAGE;
public static final int SKIP_PAGE;
public static final int EVAL_BODY_AGAIN;
public PageContext pageContext;
public Tag t;
public void setPageContext(PageContext pageContext) {
    this.pageContext=pageContext;
}
public void setParent(Tag t) {
    this.t=t;
}
public Tag getParent() {
    return t;
}
public int doStartTag()throws JspException {
    return SKIP_BODY;
}
public int doAfterBody()throws JspException {
    return SKIP_BODY;
}
public int doEndTag()throws JspException {
    return EVAL_PAGE;
}
public void release() { }
}
public class MyHandler implements TagSupport
{ ----- }

```

## -----Application7-----

### **custapp2:**

#### **iterate.jsp:**

```

<%@taglib uri="/WEB-INF/iterate.tld" prefix="mytags"%>
<mytags:iterate times="10"><br>
    Durga Software Solutions
</mytags:iterate>

```

#### **iterate.tld:**

```

<taglib>
    <jsp-version>2.1</jsp-version>
    <tlib-version>1.0</tlib-version>
    <tag>
        <name>iterate</name>
        <tag-class>com.dss.Iterate</tag-class>
        <body-content>jsp</body-content>
    </tag>
</taglib>

```

```

        <attribute>
            <name>times</name>
            <required>true</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
    </tag>
</taglib>
Iteration.java:
package com.dss;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
public class Iterate extends TagSupport
{
    int count=1;
    private int times;
    public void setTimes(int times)
    {
        this.times=times;
    }
    public int doStartTag() throws JspException
    {
        return EVAL_BODY_INCLUDE;
    }
    public int doAfterBody() throws JspException
    {
        if(count<times)
        {
            count++;
            return EVAL_BODY_AGAIN;
        }
        else
            return SKIP_BODY;
    }
}

```



## Nested Tags:

---

Defining a tag inside a tag is called as **Nested Tag**.

In custom tags application, if we declare any nested tag then we have to provide a separate configuration in tld file and we have to prepare a separate TagHandler class under classes folder.



**DURGASOFT Means JAVA**  
**JAVA Means DURGASOFT**

**Mr. Nagoor Babu** M.Tech  
Trained Lakhs of Students for Last 13 years (Realtime Expert & Java Certified)

India's No.1 JAVA Trainer

More Than 1000 Students in a Single Class Room.

Weapon of DURGASOFT

### -----Application8-----

**custapp3:**

**nested.jsp:**

```
<%@taglib uri="/WEB-INF/nested.tld" prefix="mytags"%>
<h1><center>
<mytags:if condition='<%=10>20%'>
    <mytags:true>condition is true</mytags:true>
    <mytags:false>condition is false</mytags:false>
</mytags:if>
</center></h1>
```

**nested.tld:**



```

<taglib>
  <jsp-version>2.1</jsp-version>
  <tlib-version>1.0</tlib-version>
  <tag>
    <name>if</name>
    <tag-class>com.dss.If</tag-class>
    <body-content>jsp</body-content>
    <attribute>
      <name>condition</name>
      <required>true</required>
      <rtexprvalue>true</rtexprvalue>
    </attribute>
  </tag>
  <tag>
    <name>>true</name>
    <tag-class>com.dss.True</tag-class>
    <body-content>jsp</body-content>
  </tag>
  <tag>
    <name>>false</name>
    <tag-class>com.dss.False</tag-class>
    <body-content>jsp</body-content>
  </tag>
</taglib>

```

### **If.java:**

```

package com.dss;

import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class If extends TagSupport
{
    private boolean condition;
    public void setCondition(boolean condition)
    {
        this.condition=condition;
    }
    public boolean getCondition()
    {
        return condition;
    }
    public int doStartTag() throws JspException
    {
        return EVAL_BODY_INCLUDE;
    }
}

```

### **True.java:**

```

package com.dss;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
public class True extends TagSupport
{
    public int doStartTag() throws JspException
    {
        If i=(If)getParent();
        boolean condition=i.getCondition();
        if(condition == true)
            return EVAL_BODY_INCLUDE;
        else
            return SKIP_BODY;
    }
}

```

#### **False.java:**

```

package com.dss;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
public class False extends TagSupport
{
    public int doStartTag() throws JspException
    {
        If i=(If)getParent();
        boolean condition=i.getCondition();
        if(condition == true)
            return SKIP_BODY;
        else
            return EVAL_BODY_INCLUDE;
    }
}

```

### -----Application9-----

#### **custapp4:**

#### **empdetails.jsp:**

```

<%@taglib uri="/WEB-INF/emp.tld" prefix="emp"%>
<emp:empDetails/>

```

#### **emp.tld:**

```

<taglib>
    <jsp-version>2.1</jsp-version>
    <tlib-version>1.0</tlib-version>

```

```

        <tag>
            <name>empDetails</name>
            <tag-class>com.dss.EmpDetails</tag-class>
            <body-content>empty</body-content>
        </tag>
    </taglib>

```

### **EmpDetails.java:**

```

package com.dss;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
import java.sql.*;
public class EmpDetails extends TagSupport
{
    Connection con;
    Statement st;
    ResultSet rs;
    public EmpDetails()
    {
        try
        {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","
durga");
            st=con.createStatement();
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }

    public int doStartTag() throws JspException
    {
        try
        {
            JspWriter out=pageContext.getOut();
            rs=st.executeQuery("select * from emp");
            ResultSetMetaData rsmd=rs.getMetaData();
            int count=rsmd.getColumnCount();
            out.println("<html><body bgcolor='pink'>");
            out.println("<center><br><br>");
            out.println("<table border='1' bgcolor='lightyellow'>");
            out.println("<tr>");
            for (int i=1;i<=count;i++)
            {
                out.println("<td><b><font size='6'
color='red'><center>" +rsmd.getColumnName(i) + "</center></font></b></td>");
            }
            out.println("</tr>");
            while (rs.next())
            {

```

```

        out.println("<tr>");
        for (int i=1;i<=count;i++)
        {
            out.println("<td><b><font
size='5'>"+rs.getString(i)+"</font></b></td>");
        }
        out.println("</tr>");
    }
    out.println("</table></center></body></html>");
}
catch (Exception e)
{
    e.printStackTrace();
}
return SKIP_BODY;
}
}

```

**DURGASOFT Means JAVA**  
**JAVA Means DURGASOFT**

**Mr. Nagoor Babu** M.Tech.  
 Trained Lakhs of Students for Last 13 years (Realtime Expert & Java Certified)

India's No.1 JAVA Trainer  
 More Than 1000 Students in a Single Class Room.  
 Weapon of DURGASOFT

### 3. Body Tags:

Up to now in our custom tags (simple classic tags, iteration tags) we prepared custom tags with the body, where we did not perform updations over the custom tag body, just we scanned custom tag body and displayed on the client browser.

If we want to perform updations over the custom tag body then we have to use Body Tags.

If we want to design body tags in Jsp technology then the respective TagHandler class must implement BodyTag interface either directly or indirectly.

```

public interface BodyTag extends IterationTag
{

```



```

public static final int EVAL_BODY_INCLUDE;
public static final int SKIP_BODY;
public static final int EVAL_PAGE;
public static final int SKIP_PAGE;
public static final int EVAL_BODY_AGAIN;
public static final int EVAL_BODY_BUFFERED;
public void setPageContext(PageContext pageContext);
public void setParent(Tag t);
public Tag getParent();
public int doStartTag()throws JspException;
public void doInitBody()throws JspException;
public void setBodyContent(BodyContent bodyContent);
public int doAfterBody()throws JspException;
public int doEndTag()throws JspException;
public void release();
}
public class MyHandler implements BodyTag
{ ----- }

```

In case of body tags, there are 3 possible return values from doStartTag() method.

1. EVAL\_BODY\_INCLUDE
2. SKIP\_BODY
3. EVAL\_BODY\_BUFFERED

If we return EVAL\_BODY\_INCLUDE constant from doStartTag() method then container will evaluate the custom tag body i.e. display as it is the custom tag body on client browser.

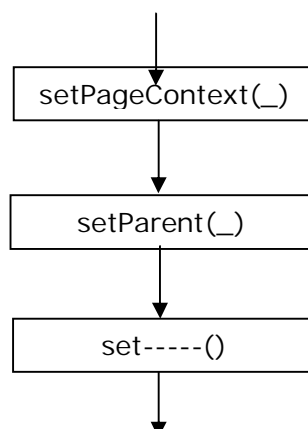
If we return SKIP\_BODY constant from doStartTag() method then container will skip the custom tag body.

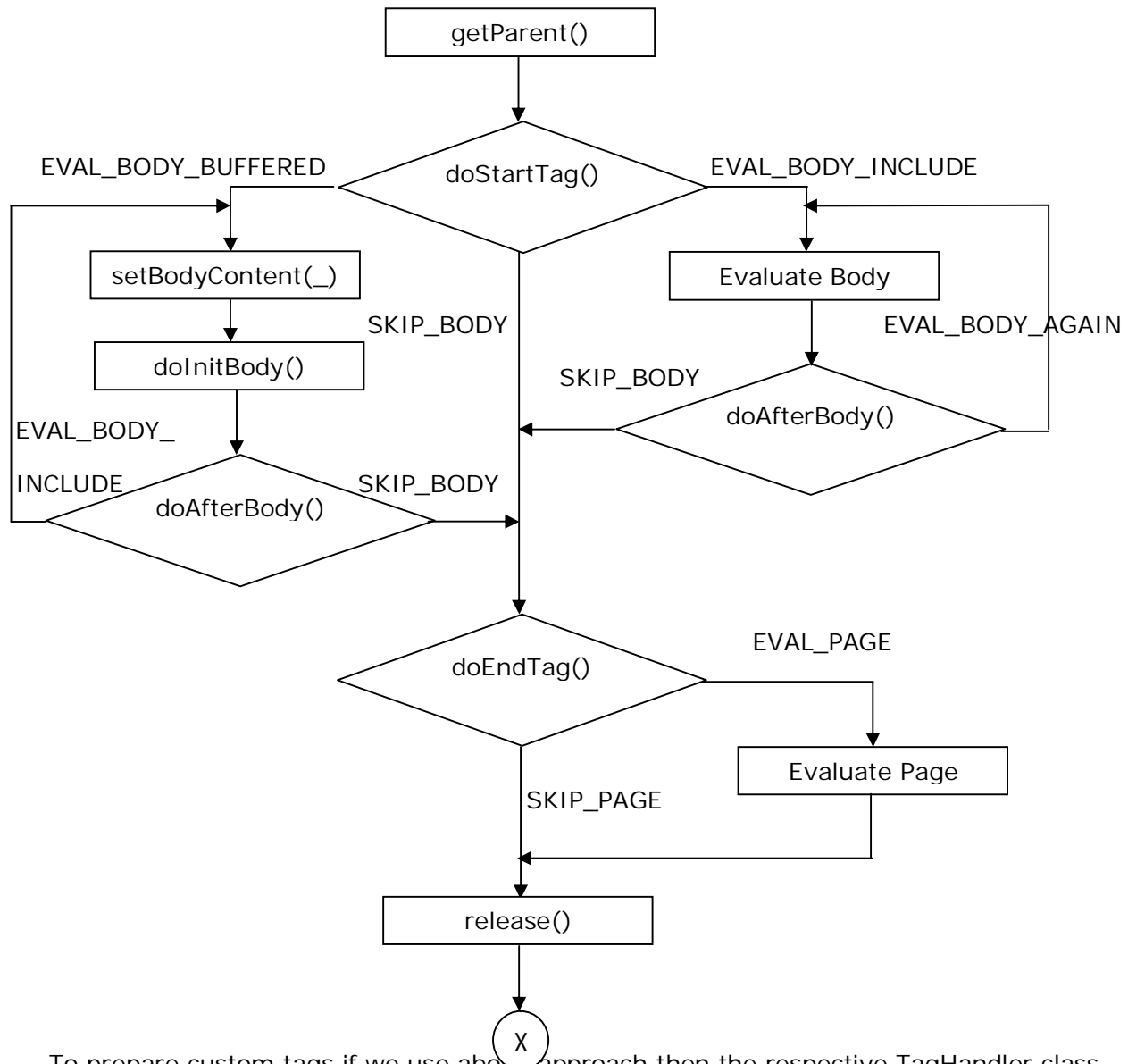
If we return EVAL\_BODY\_BUFFERED constant from doStartTag() method then container will store custom tag body in a buffer then access setBodyContent(\_) method.

To access setBodyContent(\_) method container will prepare BodyContent object with the buffer.

After executing setBodyContent(\_) method container will access doInitBody() method in order to prepare BodyContent object for allow modifications.

### Life Cycle of BodyTag interface:





To prepare custom tags if we use above approach then the respective TagHandler class must implement all the methods declared in BodyTag interface irrespective of the application requirement.

This approach may increase burden to the developers and unnecessary methods in the custom tag application.

To overcome this problem we will use an alternative provided by Jsp technology i.e. javax.servlet.jsp.tagext.BodyTagSupport class.

BodyTagSupport class is a concrete class, a direct implementation class to BodyTag interface and it has provided the default implementation for all the methods declared in BodyTag interface.

If we want to prepare custom tags with BodyTagSupport class then the respective TagHandler class must extend BodyTagSupport and overrides the only required methods.

```

public class BodyTagSupport implements BodyTag {
    public static final int EVAL_BODY_INCLUDE;
    public static final int SKIP_BODY;
    public static final int EVAL_PAGE;
    public static final int SKIP_PAGE;
    public static final int EVAL_BODY_AGAIN;
    public static final int EVAL_BODY_BUFFERED;
    public PageContext pageContext;
    public Tag t;
    public BodyContent bodyContent;
    public void setPageContext(PageContext pageContext) {
        this.pageContext=pageContext;
    }
    public void setParent(Tag t) {
        this.t=t;
    }
    public Tag getParent() {
        return t;
    }
    public int doStartTag()throws JspException {
        return EVAL_BODY_BUFFERED;
    }
    public void setBodyContent(BodyContent bodyContent) {
        this.bodyContent=bodyContent;
    }
    public void doInitBody()throws JspException
    { }
    public int doAfterBody()throws JspException {
        return SKIP_BODY;
    }
    public int doEndTag()throws JspException {
        return EVAL_PAGE;
    }
    public void release()
    { }
}

public class MyHandler extends BodyTagSupport
{ ----- }

```

In case of body tags, custom tag body will be available in BodyContent object, to get custom tag body from BodyContent object we have to use the following method.

To send modified data to the response object we have to get JspWriter object from BodyContent, for this we have to use the following method.

```

public JspWriter getEnclosingWriter()

```

# DURGASOFT Means JAVA JAVA Means DURGASOFT



**Mr. Nagoor Babu** M.Tech  
Trained Lakhs of Students for Last 13 years (Realtime Expert & Java Certified)

India's No.1 JAVA Trainer

More Than 1000 Students in a Single Class Room.

Weapon of DURGASOFT

## FREE TRAINING VIDEOS

You **Tube** **3000+ VIDEOS**

[www.youtube.com/durgasoftware](http://www.youtube.com/durgasoftware)

-----Application10-----

custapp5:

reverse.jsp:

```
<%@taglib uri="/WEB-INF/reverse.tld" prefix="mytags"%>
<mytags:reverse>
    Durga Software Solutions
</mytags:reverse>
```

reverse.tld:

DURGASOFT, # 202, 2<sup>nd</sup> Floor, HUDAMaitrivanam, Ameerpet, Hyderabad - 500038, ☎ 040 - 64 51 27 86,  
80 96 96 96 96, 9246212143 | [www.durgasoft.com](http://www.durgasoft.com)



```

<taglib>
  <jsp-version>2.1</jsp-version>
  <tlib-version>1.0</tlib-version>
  <tag>
    <name>reverse</name>
    <tag-class>com.dss.Reverse</tag-class>
    <body-content>jsp</body-content>
  </tag>
</taglib>

```

### **Reverse.java:**

```

package com.dss;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
public class Reverse extends BodyTagSupport
{
    public int doEndTag() throws JspException
    {
        try
        {
            String data=bodyContent.getString();
            StringBuffer sb=new StringBuffer(data);
            sb.reverse();
            JspWriter out=bodyContent.getEnclosingWriter();
            out.println("<html>");
            out.println("<body bgcolor='lightyellow'>");
            out.println("<center><b><font size='7' color='red'>");
            out.println("<br><br>");
            out.println(sb);
            out.println("</font></b></center></body><html>");
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
        return EVAL_PAGE;
    }
}

```

### **-----Application11-----**

### **custapp6:**

### **editor.html:**

```

<html>

<body bgcolor="lightgreen">
  <b><font size="6" color="red">
  <form action="./result.jsp">
  <pre>

```

```

        Enter SQL Query
        <textarea name="query" rows="5" cols="50"></textarea>
        <input type="submit" value="GetResult"/>
    </pre>
    </form></font></b></body>
</html>

```

### **result.jsp:**

```

<%@taglib uri="/WEB-INF/result.tld" prefix="dbtags"%>
<jsp:declaration>
    String query;
</jsp:declaration>
<jsp:scriptlet>
    query=request.getParameter("query");
</jsp:scriptlet>
<dbtags:query>
    <jsp:expression>query</jsp:expression>
</dbtags:query>

```

### **result.tld:**

```

<taglib>
    <jsp-version>2.1</jsp-version>
    <tlib-version>1.0</tlib-version>
    <tag>
        <name>query</name>
        <tag-class>com.dss.Result</tag-class>
        <body-content>jsp</body-content>
    </tag>
</taglib>

```

### **Result.java:**

```

package com.dss;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
import java.sql.*;
public class Result extends BodyTagSupport
{
    Connection con;
    Statement st;
    ResultSet rs;
    public Result()
    {
        try
        {
            Class.forName("oracle.jdbc.driver.OracleDriver");

            con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","durga");
            st=con.createStatement();
        }
    }
}

```

```

        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
    public int doEndTag() throws JspException
    {
        try
        {
            JspWriter out=pageContext.getOut();
            String query=bodyContent.getString();
            boolean b=st.execute(query);
            if (b==true)
            {
                rs=st.getResultSet();
                ResultSetMetaData rsmd=rs.getMetaData();
                out.println("<html>");
                out.println("<body bgcolor='lightblue'>");
                out.println("<center><br><br>");
                out.println("<table border='1' bgcolor='lightyellow'>");
                int count=rsmd.getColumnCount();
                out.println("<tr>");
                for (int i=1;i<=count;i++)
                {
                    out.println("<td><center><b><font size='6'
color='red'>" +rsmd.getColumnName(i) + "</font></b></center></td>");
                }
                out.println("</tr>");
                while (rs.next())
                {
                    out.println("<tr>");
                    for (int i=1;i<=count;i++)
                    {
                        out.println("<td><h1>" +rs.getString(i) + "</h1></td>");
                    }
                    out.println("</tr>");
                }
                out.println("</table></center></body></html>");
            }
            else
            {
                int rowCount=st.getUpdateCount();
                out.println("<html>");
                out.println("<body bgcolor='lightyellow'>");
                out.println("<center><b><font size='7' color='red'>");
                out.println("<br><br>");
                out.println("Record Updated : " +rowCount);

                out.println("</font></b></center></body></html>");
            }
        }
        catch (Exception e)

```

```
{  
    e.printStackTrace();  
}  
return EVAL_PAGE;  
}
```

# DURGASOFT Means JAVA JAVA Means DURGASOFT



**India's No.1 JAVA Trainer**

More Than 1000 Students in a Single Class Room.

**Weapon of DURGASOFT**

**Mr. Nagoor Babu** M.Tech  
Trained Lakhs of Students for Last 13 years (Realtime Expert & Java Certified)

## Simple Tags:

**Q: What are the differences between Classic tags and Simple tags?**

**Ans:** 1. Classic tags are more API independent, but Simple tags are less API independent.

2. If we want to design custom tags by using classic tag library then we have to remember 3 types of life cycles.

If we want to design custom tags by using simple tag library then we have to remember only 1 type of life cycle.



3. In case of classic tag library, all the TagHandler class objects are cacheable objects, but in case of simple tag library, all the TagHandler class objects are non-cacheable objects.

4. In case of classic tags, all the custom tags are not body tags by default, but in case of simple tags, all the custom tags are having body tags capacity by default.

If we want to design custom tags by using simple tag library then the respective TagHandler class must implement SimpleTag interface either directly or indirectly.

```
public interface SimpleTag extends JspTag {  
    public void setJspContext(JspContext jspContext);  
    public void setParent(JspTag t);  
    public JspTag getParent();  
    public void setJspBody(JspFragment jspFragment);  
    public void doTag()throws JspException, IOException;  
}  
public class MyHandler implements SimpleTag  
{ ----- }
```

Where jspContext is an implicit object available in simple tag library, it is same as pageContext, it can be used to make available all the Jsp implicit objects.

Where jspFragment is like bodyContent in classic tag library, it will accommodate custom tag body directly.

Where setJspContext(\_) method can be used to inject JspContext object into the present web application.

Where setParent(\_) method can be used to inject parent tags TagHandler class object reference into the present TagHandler class.

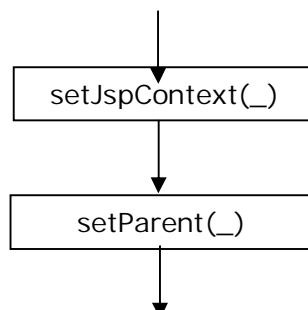
Where getParent() method can be used to get parent tags TagHandler class object.

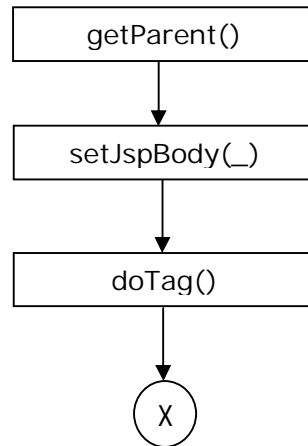
Where setJspBody(\_) method is almost all equal to setBodyContent(\_) method in order to accommodate custom tag body.

Where doTag() method is equivalent to doStartTag() method and doEndTag() method in order to perform an action.

## Life Cycle of SimpleTag interface:

---





To design custom tags if we use approach then the respective TagHandler class must implement SimpleTag interface i.e. it must implement all the methods which are declared in SimpleTag interface.

This approach will increase burden to the developers and unnecessary methods in the custom tags.

To overcome the above problem Jsp technology has provided an alternative in the form of javax.servlet.jsp.tagext.SimpleTagSupport class.

SimpleTagSupport is a concrete class provided by Jsp technology as an implementation class to SimpleTag interface with default implementation.

If we want to design custom tags by using SimpleTagSupport class then the respective TagHandler class must be extended from SimpleTagSupport class and where we have to override the required methods.

```
public interface SimpleTag extends JspTag {
    private JspContext jspContext;
    private JspFragment jspFragment;
    private JspTag jspTag;
    public void setJspContext(JspContext jspContext) {
        this.jspContext=jspContext;
    }
    public void setParent(JspTag t) {
        this.jspTag=jspTag;
    }
    public void setJspBody(JspFragment jspFragment) {
        this.jspFragment=jspFragment;
    }
    public JspTag getParent() {
        return jspTag;
    }
    public JspFragment getJspBody() {
        return jspFragment;
    }
    public JspContext getJspContext() {
        return jspContext;
    }
}
```

```

public void doTag()throws JspException, IOException
{
}
}
public class MyHandler extends SimpleTagSupport
{
-----
}

```

## DURGASOFT Means JAVA JAVA Means DURGASOFT





**India's No.1 JAVA Trainer**

More Than 1000 Students in a Single Class Room.

**Weapon of DURGASOFT**

**Mr. Nagoor Babu** M.Tech  
Trained Lakhs of Students for Last 13 years (Realtime Expert & Java Certified)

LEARN FROM EXPERT & DIAMOND FACULTIES OF AMEERPET...

# JAVA MEANS DURGASOFT

INDIA'S NO. 1 SOFTWARE TRAINING INSTITUTE

AN ISO 9001:2008 CERTIFIED  
**DURGA**  
SOFTWARE SOLUTIONS

#202 2<sup>nd</sup> FLOOR  
www.durgasoft.com

040-64512786  
+91 9246212143  
+91 8096969696

-----Application12-----

**custapp7:**  
**hello.jsp:**

```

<%@taglib uri="/WEB-INF/hello.tld" prefix="mytags"%>
<mytags:hello/>

```

**hello.tld:**

```

<taglib>
  <jsp-version>2.1</jsp-version>
  <tlib-version>1.0</tlib-version>

```

```
<tag>
    <name>hello</name>
    <tag-class>com.dss.Hello</tag-class>
    <body-content>empty</body-content>
</tag>
</taglib>
```

**Hello.java:**

```
package com.dss;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
import java.io.*;
public class Hello extends SimpleTagSupport
{
    public void doTag() throws JspException,IOException
    {
        getJspContext().getOut().println("<h1><center>Hello SimpleTag
Application</center></h1>");
    }
}
```





*LEARN FROM EXPERTS ...*

# COMPLETE JAVA

CORE JAVA, ADV. JAVA, ORACLE, STRUTS, HIBERNATE, SPRING, WEB SERVICES,...

# COMPLETE .NET

C#.NET, ASP.NET, SQL SERVER, MVC 5 & WCF

# TESTING TOOLS

MANUAL + SELENIUM

# ORACLE | D2K

# MSBI | SHARE POINT

# HADOOP | ANDROID

# C, C++, DS, UNIX

# CRT & APTITUDE TRAINING

AN ISO 9001:2008 CERTIFIED

**DURGA**

Software Solutions®

# 202, 2nd Floor, HUDA Maitrivanam,  
Ameerpet, Hyd. Ph: 040-64512786,

**9246212143, 8096969696**

**[www.durgasoft.com](http://www.durgasoft.com)**