Custom Taglibs

Creating Tag Libraries

garth@ggilmour.com

Tag Libraries

- The JSP spec defines an API for creating Actions
  - The classes involved live in ‘javax.servlet.jsp.tagext’
    - An action class inherits from ‘TagSupport’ or ‘BodyTagSupport’
  - When the JSP Engine meets an action element:
    - An instance of the corresponding class is created
    - Callback methods of the class are invoked
- Actions are grouped together into ‘Tag Libraries’
  - Third party frameworks like Struts use them heavily
  - Content management frameworks use them for customization
  - The JSTL groups the most frequently used actions
    - Such as selection, iteration, using JDBC and manipulating XML

Introducing TagUnit

- TagUnit is a unit testing tool for Tag Libraries
  - Just as JUnit tests conventional classes and HttpUnit enables you to test Servlets and JavaServer Pages
- TagUnit provides you with:
  - A web application that automatically runs tests
  - A Tag Library containing actions to let you test other tags
- The benefits of TagUnit are:
  - Development is easier and faster (‘big red button’)
  - The tests can be used to help maintain the code
  - Other developers can use the tests to learn your library

Installing TagUnit

- To add TagUnit to a web application:
  - Copy ‘tagunit.jar’ into ‘WEB-INF/lib’
  - Copy the folder ‘tests’ into the root directory
    - This contains all the resources required by the GUI
  - Configure the TagUnit controller servlet in ‘web.xml’
    - See the example on the following slide
- You should have a separate web app for testing
  - If you are deploying to other teams include the tests
  - TagUnit comes with a startup app called ‘tagunit-blank’
  - There is also ‘tagunit-examples’ which unit tests the JSTL

TagUnit Configuration in ‘web.xml’

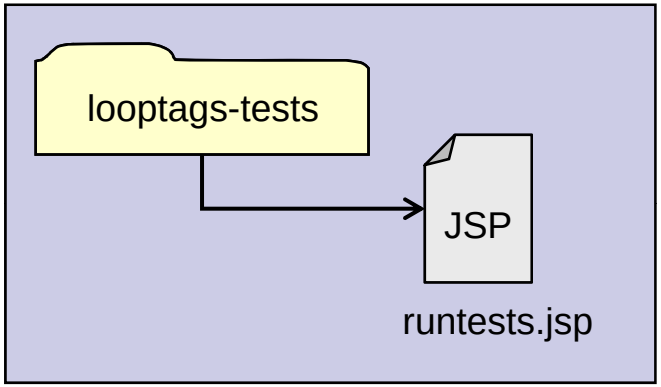
```
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
    version="2.4">

    <servlet>
        <servlet-name>TagUnitTestController</servlet-name>
        <servlet-class>org.tagunit.controller.FrontController</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>TagUnitTestController</servlet-name>
        <url-pattern>/test/servlet/*</url-pattern>
    </servlet-mapping>
</web-app>
```

Using TagUnit

- To run the basic sanity tests provided by TagUnit:
  - Create a folder in your web app named after the library
  - Inside the folder create a JSP which points TagUnit at the Tag Library to be tested
    - Using the ‘testTagLibrary’ and ‘tagLibraryDescriptor’ actions
  - Run TagUnit by calling the controller Servlet
    - The path to your JSP is passed as a parameter
- To add your own unit tests
  - Create one subfolder for each tag to test
  - Inside the subfolder put JSP’s which:
    - Have names beginning with ‘test’
    - Use the TagUnit assertion actions
      - These are modelled on JUnit assertions

Setup for Unit Tests



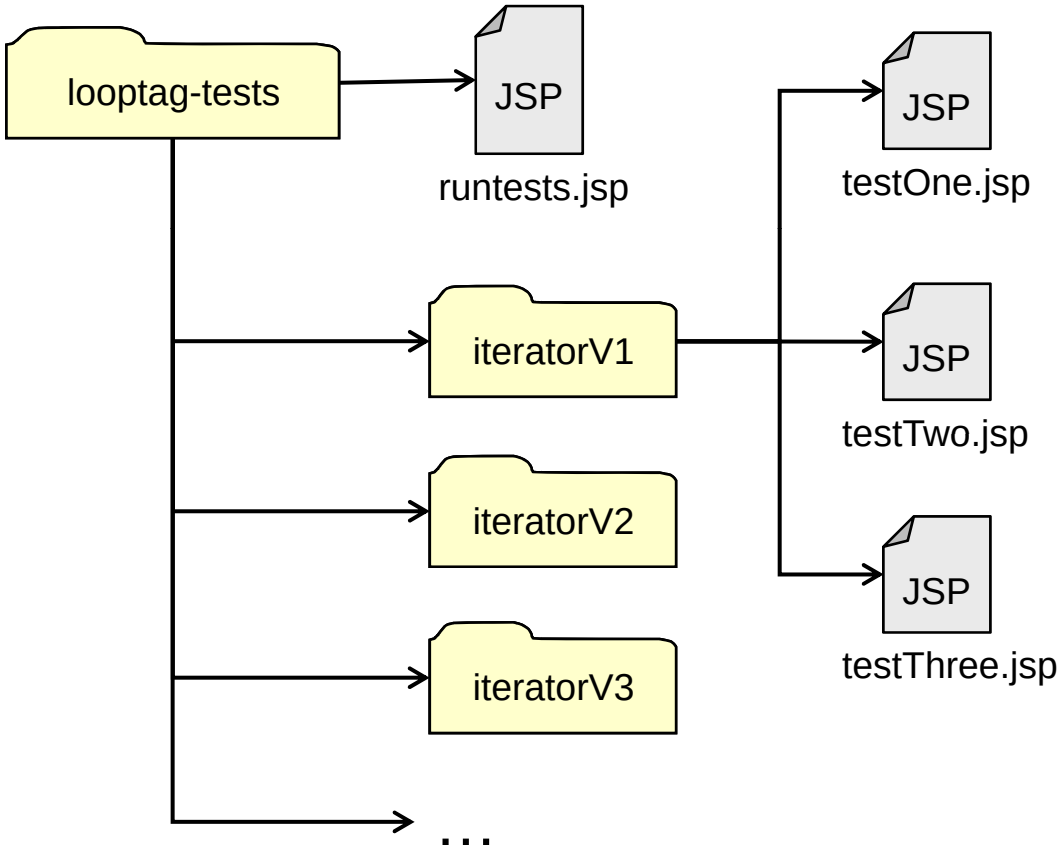
Contents of ‘runtests.jsp’

```
<%@ taglib uri="http://www.tagunit.org/tagunit/core" prefix="tagunit"%>
<tagunit:testTagLibrary uri="/looptags-tests">
    <tagunit:tagLibraryDescriptor uri="/WEB-INF/tlds/iterators.tld"/>
</tagunit:testTagLibrary>
```

Link required to trigger unit tests

```
<a href="/actions/test/servlet/RunTests?uri=/looptags-tests/runtests.jsp">Run Tests</a>
```

Writing Unit Tests



Tag Libraries

- The TLD file for a Tag Library contains
  - Versioning information about the standards in use
  - A description of each tag
    - Including its name, content model, attributes and implementation
    - Attributes are defined in terms of their name, whether they are optional and if their value can be determined at runtime
- Some Tags are more complex than others:
  - Tags with attributes but no content are the simplest
  - Tags with content have more complex state transitions
  - Tags which contain other actions use an object hierarchy
  - Tags which set JSP attributes need an additional helper class

A Simple TLD

```
<!DOCTYPE taglib PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
    "http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">
<taglib>
    <tlib-version>1.2</tlib-version>
    <jsp-version>1.2</jsp-version>
    <short-name>my library</short-name>
    <tag>
        <name>MyTag</name>
        <tag-class>demos.actions.MyTagClass</tag-class>
        <body-content>empty</body-content>
        <attribute>
            <name>att1</name>
            <required>true</required>
            <rtexprvalue>false</rtexprvalue>
        </attribute>
    </tag>
</taglib>
```

Listeners and Tag Libraries

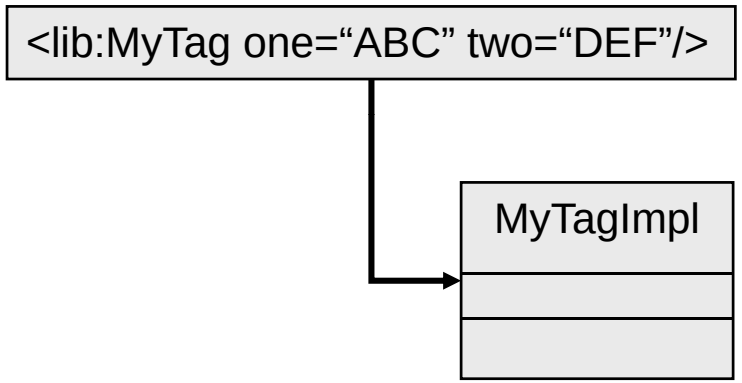
- Listener interfaces let arbitrary code react to events
  - Implementing classes must have a no-args constructor
- Listeners can be registered in TLD's as well as 'web.xml'
  - Those added in TLD's are treated as extensions of those in 'web.xml'
  - The listeners of multiple tag libraries are registered in an unpredictable order, but all are registered before the application starts

Interface Name	Purpose
ServletContextListener	React to creation and destruction of the context object
HttpSessionListener	React to creation and destruction of sessions
ServletRequestListener	React to creation and destruction of requests
ServletContextAttributeListener	React to attributes being manipulated in the context
HttpSessionAttibuteListener	React to attributes being manipulated in sessions
ServletRequestAttributeListener	React to attributes being manipulated in requests

Creating Simple Tags

- To create an attributes only action:
  - Create a TLD file and describe your action
  - Link your JSP to the TLD via a ‘taglib’ directive
  - Write and compile the implementation class
- The implementation class should:
  - Inherit from the base class ‘TagSupport’
  - Define JavaBean setter methods for each attribute
  - Override one of the lifecycle methods
    - ‘doStartTag’ is called at the start of the action
    - ‘doEndTag’ is called at the end of the action
    - Both return a constant representing how processing should continue

Processing Simple Tags




```
MyTagImpl tag = new MyTagImpl();
tag.setPageContext(pageContext);
tag.setParent(null);
tag.setOne("ABC");
tag.setTwo("DEF");
tag.doStartTag();
if (tag.doEndTag() == Tag.SKIP_PAGE) {
      //Stop processing page
}
//Keep processing page
```

Creating Simple Tags

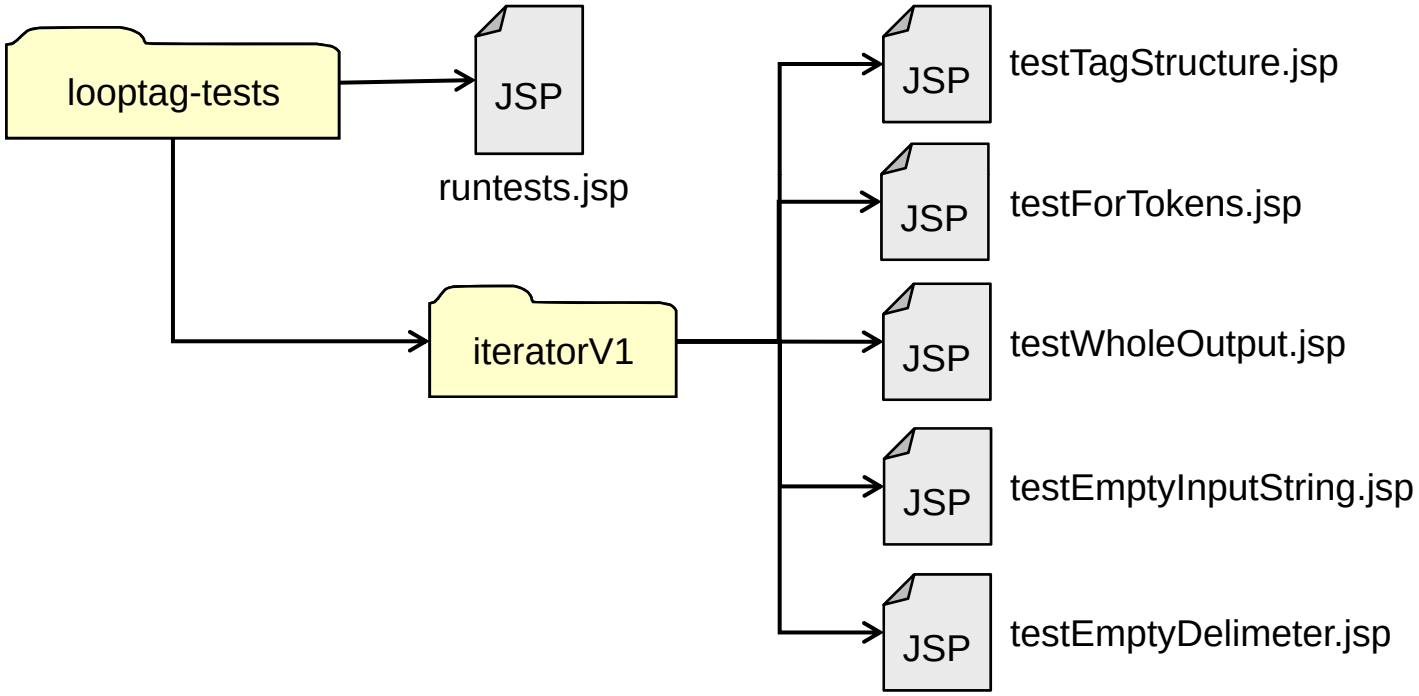
```
<iter:iteratorV1 string="A#B#C#D#E" delim="#"/>
```

```
<tag>
    <name>iteratorV1</name>
    <tag-class>looptags.v1.Iterator</tag-class>
    <body-content>empty</body-content>
    <attribute>
        <name>string</name>
        <required>true</required>
        <rtexprvalue>false</rtexprvalue>
    </attribute>
    <attribute>
        <name>delim</name>
        <required>true</required>
        <rtexprvalue>false</rtexprvalue>
    </attribute>
</tag>
```

Creating Simple Tags

```
package looptags.v1;
//import statements omitted
public class Iterator extends TagSupport {
	public void setString(String string) { this.string = string; }
	public void setDelim(String delim) {  this.delim = delim; }
	public int doEndTag() {
		try {
			if(null == string || string.equals("")) {
				pageContext.getOut().println("<p>Nothing to iterate over</p>");
			} else {
				StringTokenizer st = new StringTokenizer(string,delim);
				while(st.hasMoreTokens()) {
					pageContext.getOut().println("<p>" + st.nextToken() + "</p>");
				}
			}
		} catch(IOException ex) {  System.out.println("\tERROR " + ex);  }
		return EVAL_PAGE;
	}
	private String string;
	private String delim;
}
```

Unit Testing IteratorV1



Unit Tests for IteratorV1

testTagStructure.jsp

```
<%@ taglib uri="http://www.tagunit.org/tagunit/core" prefix="tagunit"%>
<%@taglib uri="/WEB-INF/tlds/iterators.tld" prefix="iter" %>

<tagunit:assertAttribute name="string" required="true" rtexprvalue="false"/>
<tagunit:assertAttribute name="delim" required="true" rtexprvalue="false"/>
<tagunit:assertBodyContent name="empty"/>
```

Unit Tests for IteratorV1

testEmptyInputString.jsp

```
<%@ taglib uri="http://www.tagunit.org/tagunit/core" prefix="tagunit"%>
<%@taglib uri="/WEB-INF/tlds/iterators.tld" prefix="iter" %>

<tagunit:assertEquals name="Test Output For Empty Input String"
                      ignoreWhitespace="true">

    <tagunit:actualResult>
        <iter:iteratorV1 string="" delim="#"/>
    </tagunit:actualResult>
    <tagunit:expectedResult>
        <p>Nothing to iterate over</p>
    </tagunit:expectedResult>
</tagunit:assertEquals>
```

Unit Tests for IteratorV1

testEmptyDelimeter.jsp

```
<%@ taglib uri="http://www.tagunit.org/tagunit/core" prefix="tagunit"%>
<%@taglib uri="/WEB-INF/tlds/iterators.tld" prefix="iter" %>

<tagunit:assertEquals name="Test Output For Empty Delimeter"
                      ignoreWhitespace="true">

    <tagunit:actualResult>
        <iter:iteratorV1 string="A#B#C#D#E" delim=""/>
    </tagunit:actualResult>
    <tagunit:expectedResult><p>A#B#C#D#E</p></tagunit:expectedResult>
</tagunit:assertEquals>
```

Unit Tests for IteratorV1

testForTokens.jsp

```
<%@ taglib uri="http://www.tagunit.org/tagunit/core" prefix="tagunit"%>
<%@taglib uri="/WEB-INF/tlds/iterators.tld" prefix="iter" %>

<tagunit:assertContains name="Has first token">
    <tagunit:actualResult>
        <iter:iteratorV1 string="A#B#C#D#E" delim="#"/>
    </tagunit:actualResult>
    <tagunit:expectedResult>A</tagunit:expectedResult>
</tagunit:assertContains>

<tagunit:assertContains name="Has second token">
    <tagunit:actualResult>
        <iter:iteratorV1 string="A#B#C#D#E" delim="#"/>
    </tagunit:actualResult>
    <tagunit:expectedResult>B</tagunit:expectedResult>
</tagunit:assertContains>
<%-- Assertions continued for other three tokens --%>
```

Unit Tests for IteratorV1

testWholeOutput.jsp

```
<%@ taglib uri="http://www.tagunit.org/tagunit/core" prefix="tagunit"%>
<%@taglib uri="/WEB-INF/tlds/iterators.tld" prefix="iter" %>

<%
    final String expectedResult="<p>A</p><p>B</p><p>C</p><p>D</p><p>E</p>";
%>

<tagunit:assertEquals name="Test Output" ignoreWhitespace="true">
    <tagunit:actualResult>
        <iter:iteratorV1 string="A#B#C#D#E" delim="#"/>
    </tagunit:actualResult>
    <tagunit:expectedResult><%= expectedResult %></tagunit:expectedResult>
</tagunit:assertEquals>
```

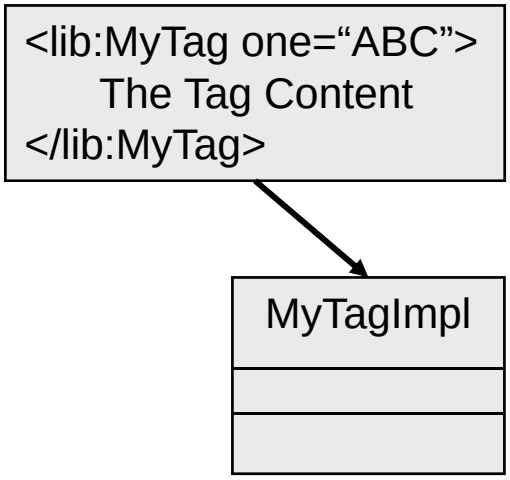
Processing the Tag Body

- Tags which contain content should:
  - Set their ‘body-content’ element to a value other than ‘empty’
    - The choices are either ‘JSP’ or ‘tagdependant’
  - Inherit from ‘BodyTagSupport’
- The ‘doStartTag’ method can return three values:
  - ‘EVAL_BODY_INCLUDE’
    - Includes the body in the current output stream
  - ‘EVAL_BODY_BUFFERED’
    - Creates a ‘BodyContent’ object to contain the evaluated body
    - This is the value returned by the base class method
  - ‘SKIP_BODY’
    - Ignores the body completely

Processing the Tag Body

- ‘getBodyContent’ returns the ‘BodyContent’ object
  - The ‘BodyContent’ class inherits from ‘JspWriter’
  - It has a ‘getString’ method for reading short content
- The object contains the evaluated content of the tag
  - Any code or nested tags are executed and their output included
- Extra callbacks are available for Tags with a body
  - ‘doInitBody’ is called before the body is read for the first time
  - ‘doAfterBody’ is called each time the body is evaluated
    - We will use this method later to create a more complex loop

Processing Tags With Content




```
MyTagImpl tag = new MyTagImpl();
tag.setPageContext(pageContext);
tag.setParent(null);
tag.setOne("ABC");
int retval = tag.doStartTag();
if(retval != Tag.SKIP_BODY) {
   if(retval != Tag.EVAL_BODY_INCLUDE) {
        BodyContent bc = new BodyContent();
        tag.setBodyContent(bc);
        tag.doInitBody();
        do {
             //Process body into BodyContent object
        } while(BodyTag.EVAL_BODY_AGAIN == tag.doAfterBody());
   }
}
if (tag.doEndTag() == Tag.SKIP_PAGE) {
   //Stop processing page
}
//Keep processing page
```

Processing the Tag Body

```
<iter:iteratorV2 delim="#">A#B#C#D#E</iter:iteratorV2>
```

```
<tag>
    <name>iteratorV2</name>
    <tag-class>looptags.v2.Iterator</tag-class>
    <body-content>JSP</body-content>
    <attribute>
        <name>delim</name>
        <required>true</required>
        <rtexprvalue>false</rtexprvalue>
    </attribute>
</tag>
```

Processing the Tag Body

```
package looptags.v2;
//import statements omitted

public class Iterator extends BodyTagSupport {
	public void setDelim(String delim) { this.delim = delim; }
	public int doEndTag() {
		BodyContent body = getBodyContent();
		try {
			if(null == body || null == body.getString()) {
				pageContext.getOut().println("<p>Nothing to iterate over</p>");
			} else {
				StringTokenizer st = new StringTokenizer(body.getString(),delim);
				while(st.hasMoreTokens()) {
					pageContext.getOut().println("<p>" + st.nextToken() + "</p>");
				}
			}
		} catch(IOException ex) { System.out.println("\tERROR " + ex); }
		return EVAL_PAGE;
	}
	private String delim;
}
```

Special Tests for IteratorV2

testTagStructure.jsp

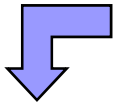
```
<%@ taglib uri="http://www.tagunit.org/tagunit/core" prefix="tagunit"%>
<%@taglib uri="/WEB-INF/tlds/iterators.tld" prefix="iter" %>

<tagunit:assertAttribute name="delim" required="true" rtexprvalue="false"/>
<tagunit:assertBodyContent name="JSP"/>
```

Nesting Actions

- Nested actions do not exist inside ‘BodyContent’ objects
  - They are evaluated and their result is stored in the stream
- Nested actions can communicate with their ‘parent’ action via its accessor methods
  - The ‘TagSupport.getParent’ method allows an action to obtain a reference to the enclosing tag
    - The interaction is them the same as any two JavaBeans
  - The ‘TagSupport.findAncestorWithClass’ method allows an action to search multiple ancestors
    - The parameters are the action to begin with and the class name of the action to find (the closest ancestor is found)

Nesting Actions



```
<iter:iteratorV3>
    <iter:delim>#</iter:delim>
    <iter:string>A#B#C#D#E</iter:string>
</iter:iteratorV3>
```

```
<tag>
    <name>iteratorV3</name>
    <tag-class>looptags.v3.Iterator</tag-class>
    <body-content>JSP</body-content>
</tag>
<tag>
    <name>delim</name>
    <tag-class>looptags.v3.IteratorDelim</tag-class>
    <body-content>JSP</body-content>
</tag>
<tag>
    <name>string</name>
    <tag-class>looptags.v3.IteratorString</tag-class>
    <body-content>JSP</body-content>
</tag>
```

The Delim Tag

```
package looptags.v3;

import javax.servlet.jsp.tagext.*;

public class IteratorDelim extends BodyTagSupport {
	public int doEndTag() {
		if(null == getBodyContent()) {
			throw new IllegalArgumentException("Delimeter must have content");
		}
		String delim = getBodyContent().getString();
		Tag parent = getParent();
		Iterator iterator = (Iterator)parent;
		iterator.setDelim(delim);
		return EVAL_PAGE;
	}
}
```

The String Tag

```
package looptags.v3;

import javax.servlet.jsp.tagext.*;

public class IteratorString extends BodyTagSupport {
	public int doEndTag() {
		if(null == getBodyContent()) {
			throw new IllegalArgumentException("String must have content");
		}
		String string = getBodyContent().getString();
		Tag parent = getParent();
		Iterator iterator = (Iterator)parent;
		iterator.setString(string);
		return EVAL_PAGE;
	}
}
```

The IteratorV3 Tag

```
package looptags.v3;
//imports omitted
public class Iterator extends BodyTagSupport {
	public void setDelim(String delim) { this.delim = delim; }
	public void setString(String string) { this.string = string; }

	public int doEndTag() {
		StringTokenizer st = new StringTokenizer(string,delim);
		try {
			while(st.hasMoreTokens()) {
				pageContext.getOut().println("<p>" + st.nextToken() + "</p>");
			}
		} catch(IOException ex) {
			System.out.println("\tERROR " + ex);
		}
		return EVAL_PAGE;
	}
	private String string;
	private String delim;
}
```

Special Tests for IteratorV3

testTagStructure.jsp

```
<%@ taglib uri="http://www.tagunit.org/tagunit/core" prefix="tagunit"%>
<%@taglib uri="/WEB-INF/tlds/iterators.tld" prefix="iter" %>

<tagunit:assertNoAttributes/>
<tagunit:assertBodyContent name="JSP"/>
```

Special Tests for IteratorV3

testEmptyInputString.jsp

```
<%@ taglib uri="http://www.tagunit.org/tagunit/core" prefix="tagunit"%>
<%@taglib uri="/WEB-INF/tlds/iterators.tld" prefix="iter" %>

<tagunit:assertException name="Exception Thrown On Empty String"
                         exception="java.lang.IllegalArgumentException"
                         message="String must have content">
    <iter:iteratorV3>
        <iter:delim>#</iter:delim>
        <iter:string></iter:string>
    </iter:iteratorV3>
</tagunit:assertException>
```

Special Tests for IteratorV3

testEmptyDelimeter.jsp

```
<%@ taglib uri="http://www.tagunit.org/tagunit/core" prefix="tagunit"%>
<%@taglib uri="/WEB-INF/tlds/iterators.tld" prefix="iter" %>

<tagunit:assertException name="Exception Thrown On Empty Delimeter"
                         exception="java.lang.IllegalArgumentException"
                         message="Delimeter must have content">
    <iter:iteratorV3>
        <iter:delim></iter:delim>
        <iter:string>A#B#C#D#E</iter:string>
    </iter:iteratorV3>
</tagunit:assertException>
```

Interacting with the JSP

- Actions can use the ‘PageContext’ of their JSP
  - You inherit a field called ‘pageContext’ from ‘TagSupport’
- ‘PageContext’ lets you use the entire JSP API
  - ‘include’ and ‘forward’ perform request dispatching
  - ‘getAttribute’, ‘setAttribute’ and ‘removeAttribute’ access attributes in any of the standard scopes
  - ‘findAttribute’ searches all the standard scopes in order
- Actions which are not nested use attributes to interact
  - Make sure the names you choose will not conflict with other developers attributes and use the most local scope possible

Interacting with the JSP

```
<tag>
	<name>setDetails</name>
	<tag-class>looptags.v4.IteratorSetDetails</tag-class>
	<body-content>empty</body-content>
	<attribute>
		<name>string</name>
		<required>true</required>
		<rtexprvalue>false</rtexprvalue>
	</attribute>
	<attribute>
		<name>delim</name>
		<required>true</required>
		<rtexprvalue>false</rtexprvalue>
	</attribute>
</tag>
<tag>
	<name>iteratorV4</name>
	<tag-class>looptags.v4.Iterator</tag-class>
	<body-content>empty</body-content>
</tag>
```

```
<iter:setDetails string="A#B#C#D#E" delim="#"/>
<iter:iteratorV4/>
```

The Set Details Tag

```
package looptags.v4;
//imports omitted
public class IteratorSetDetails extends BodyTagSupport {
	public void setString(String string) { this.string = string; }
	public void setDelim(String delim) { this.delim = delim; }
	public int doEndTag() {
		if(null == string || string.equals("")) {
			setAttributes("","Nothing to iterate over");
		} else {
			setAttributes(delim,string);
		}
		return EVAL_PAGE;
	}
	private void setAttributes(String a1, String a2) {
		pageContext.setAttribute("looptags.v4.delim",a1,PageContext.PAGE_SCOPE);
		pageContext.setAttribute("looptags.v4.string",a2,PageContext.PAGE_SCOPE);
	}
	private String string;
	private String delim;
}
```

The IteratorV4 Tag

```
package looptags.v4;
//imports omitted
public class Iterator extends BodyTagSupport {
	public int doEndTag() {
		int scope = PageContext.PAGE_SCOPE;
		String string = (String)pageContext.getAttribute("looptags.v4.string",scope);
		String delim = (String)pageContext.getAttribute("looptags.v4.delim",scope);

		StringTokenizer st = new StringTokenizer(string,delim);
		try {
			while(st.hasMoreTokens()) {
				pageContext.getOut().println("<p>" + st.nextToken() + "</p>");
			}
		} catch(IOException ex) {
			System.out.println("\tERROR " + ex);
		}
		return EVAL_PAGE;
	}
}
```

Adding Variables to the JSP

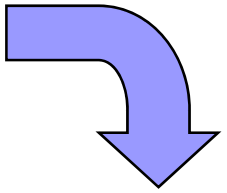
- Some actions need extra declarations in '_jspService'
  - An iterator tag that nests scriptlets needs a 'current' variable
  - The 'useBean' action needs a variable that points to the bean
- The JSP Engine adds these declarations for the action
  - The information to be added can be specified in a helper class or hardcoded inside the TLD
- The declaration name can be taken from the action
  - Most actions that add declarations let the tag user pick the name
  - The convention is that an 'id' attribute names the variable
- The 'TagData' class contains a table of attributes
  - It has an entry for each of the attributes used on the action

Adding Variables via TagExtraInfo

- Variables can be added to a JSP via a helper class
  - This class must inherit from ‘TagExtraInfo’
  - It must be configured in the TLD using ‘<teiclass>’
- Your class must override the ‘getVariableInfo’ method
  - ‘VariableInfo[ ] getVariableInfo(TagData data)’
- The container uses the object array to create variables
  - A ‘VariableInfo’ object is returned for each new declaration
  - The ‘TagData’ parameter holds attributes passed to the action
    - This lets you name variables according to the clients needs
    - For example ‘<MyTag id=“myval” val=“xyz”/>’ could create a variable called ‘myval’ with initial value ‘xyz’

Adding Variables Via Code

```
public VariableInfo[] getVariableInfo(TagData data) {
      VariableInfo[] tmparray = new VariableInfo[1];
      VariableInfo var = new VariableInfo("delim",
                                          "java.lang.String",
                                          true,
                                          VariableInfo.AT_END);
      tmparray[0] = var;
      return tmparray;
}
```



```
//Inserted on the generated JSP Servlet by Jasper
java.lang.String delim = null;
delim = (java.lang.String) pageContext.findAttribute("delim");
```

Adding Variables in the TLD

- Variable declarations can also be specified by nesting a ‘variable’ element inside ‘tag’
  - If you do this you cannot also use an extra info class

Attribute	Description
name-given	Defines the name to the variable to be created
name-from-attribute	Declares that the name of the variable will be the same as the given attribute
variable-class	The class name of the variable, default is string
scope	Scope of the variable, can be AT_BEGIN, AT_END or NESTED (the default)
declare	Whether the variable is declared after the tag invocation, defaults to true
description	The description of the variable, no default

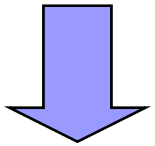
Creating Iteration Tags

- Creating iteration tags is fairly simple
  - But requires use of all the concepts we have discussed
- To create an iterator tag
  - Inside ‘doStartTag’ set up the collection to be iterated over and return ‘EVAL_BODY_INCLUDE’
    - An attribute should be set with the value of the first element
    - This should be added to the JSP page via an extra info class
  - Inside ‘doAfterBody’ check if there are more elements
    - If there are more elements set the attribute to represent the next element and return ‘EVAL_BODY_AGAIN’
    - Otherwise return ‘SKIP_BODY’

Creating Iteration Tags

```
<iter:iteratorV5 string="A#B#C#D#E" delim="#" id="current">
    Current Token is: <%= current%><br/>
</iter:iteratorV5>
```

```
<iter:iteratorV6 string="A#B#C#D#E" delim="#" id="current">
    Current Token is: <%= current%><br/>
</iter:iteratorV6>
```



```
Current Token is: A <br/>
Current Token is: B <br/>
Current Token is: C <br/>
Current Token is: D <br/>
Current Token is: E <br/>
```

Creating Iteration Tags

```
<tag>
	<name>iteratorV5</name>
	<tag-class>looptags.v5.Iterator</tag-class>
	<tei-class>looptags.v5.IteratorExtraInfo</tei-class>
	<body-content>JSP</body-content>
	<attribute>
		<name>string</name>
		<required>true</required>
		<rtexprvalue>false</rtexprvalue>
	</attribute>
	<attribute>
		<name>delim</name>
		<required>true</required>
		<rtexprvalue>false</rtexprvalue>
	</attribute>
	<attribute>
		<name>id</name>
		<required>true</required>
		<rtexprvalue>false</rtexprvalue>
	</attribute>
</tag>
```

```
<tag>
	<name>iteratorV6</name>
	<tag-class>looptags.v6.Iterator</tag-class>
	<body-content>JSP</body-content>
	<variable>
		<name-from-attribute>id</name-from-attribute>
		<variable-class>java.lang.String</variable-class>
		<declare>true</declare>
	</variable>
	<attribute>
		<name>string</name>
		<required>true</required>
		<rtexprvalue>false</rtexprvalue>
	</attribute>
	<attribute>
		<name>delim</name>
		<required>true</required>
		<rtexprvalue>false</rtexprvalue>
	</attribute>
	<attribute>
		<name>id</name>
		<required>true</required>
		<rtexprvalue>false</rtexprvalue>
	</attribute>
</tag>
```

The IteratorV5 Tag Part 1

```
package looptags.v5;

//imports omitted
public class Iterator extends BodyTagSupport {
	public int doStartTag() {
		if(null == string || string.equals("")) {
			string = "Nothing to iterate over";
			delim = "";
		}
		if(st == null) {
			st = new StringTokenizer(string,delim);
		}
		//set up the current token as we enter the loop
		if(st.hasMoreTokens()) {
			pageContext.setAttribute(id,st.nextToken());
			return EVAL_BODY_INCLUDE;
		} else {
			return SKIP_BODY;
		}
	}
```

The IteratorV5 Tag Part 2

```
public int doAfterBody() {
        //keep going back over the body while there
        // are more tokens to be processed
        if(st.hasMoreTokens()) {
                pageContext.setAttribute(id,st.nextToken());
                return EVAL_BODY_AGAIN;
        } else {
                return SKIP_BODY;
        }
}
public int doEndTag() {
        st = null;
        return EVAL_PAGE;
}
public void setString(String string) {
        this.string = string;
}
public void setId(String id) {
        this.id = id;
}
```

The IteratorV5 Tag Part 3

```
        public void setDelim(String delim) {
                this.delim = delim;
        }
        private String string;
        private String delim;
        private String id;
        private StringTokenizer st;
}
```

```
package looptags.v5;
import javax.servlet.jsp.tagext.*;
public class IteratorExtraInfo extends TagExtraInfo {
        public VariableInfo[] getVariableInfo(TagData data) {
                VariableInfo[] tmparray = new VariableInfo[1];
                VariableInfo var = new VariableInfo((String)data.getAttribute("id"),
                                                        "java.lang.String",true,VariableInfo.NESTED);
                tmparray[0] = var;
                return tmparray;
        }
}
```

Tests For IteratorV5

testTagStructure.jsp

```
<%@ taglib uri="http://www.tagunit.org/tagunit/core" prefix="tagunit"%>
<%@taglib uri="/WEB-INF/tlds/iterators.tld" prefix="iter" %>

<tagunit:assertAttribute name="string" required="true" rtexprvalue="false"/>
<tagunit:assertAttribute name="delim" required="true" rtexprvalue="false"/>
<tagunit:assertAttribute name="id" required="true" rtexprvalue="false"/>
<tagunit:assertBodyContent name="JSP"/>
```

Tests For IteratorV5

testEmptyInputString.jsp

```
<%@ taglib uri="http://www.tagunit.org/tagunit/core" prefix="tagunit"%>
<%@taglib uri="/WEB-INF/tlds/iterators.tld" prefix="iter" %>

<tagunit:assertEquals name="Test Output For Empty Input String"
                              ignoreWhitespace="true">
    <tagunit:actualResult>
        <iter:iteratorV5 string="" delim="#" id="current">
        Current Token is: <%= current%><br/>
        </iter:iteratorV5>        </tagunit:actualResult>
    <tagunit:expectedResult>
        Current Token is: Nothing to iterate over<br/>
    </tagunit:expectedResult>
</tagunit:assertEquals>
```

Tests For IteratorV5

testEmptyDelimeter.jsp

```
<%@ taglib uri="http://www.tagunit.org/tagunit/core" prefix="tagunit"%>
<%@taglib uri="/WEB-INF/tlds/iterators.tld" prefix="iter" %>

<tagunit:assertEquals name="Test Output For Empty Delimeter"
                      ignoreWhitespace="true">
    <tagunit:actualResult>
        <iter:iteratorV5 string="A#B#C#D#E" delim="" id="current">
        Current Token is: <%= current%><br/>
        </iter:iteratorV5>
    </tagunit:actualResult>
    <tagunit:expectedResult>
        Current Token is: A#B#C#D#E<br/>
    </tagunit:expectedResult>
</tagunit:assertEquals>
```

Tests For IteratorV5

testWholeOutput.jsp

```
<%@ taglib uri="http://www.tagunit.org/tagunit/core" prefix="tagunit"%>
<%@taglib uri="/WEB-INF/tlds/iterators.tld" prefix="iter" %>

<%
        final String expectedResult="Current Token is: A<br/>”;
        expectedResult += “Current Token is: B<br/>”;
        expectedResult +=“Current Token is: C<br/>”;
        expectedResult +=“Current Token is: D<br/>”;
        expectedResult +=“Current Token is: E<br/>";
%>

<tagunit:assertEquals name="Test Output" ignoreWhitespace="true">
        <tagunit:actualResult>
                <iter:iteratorV5 string="A#B#C#D#E" delim="#" id="current">
                Current Token is: <%= current%><br/>
                </iter:iteratorV5>
        </tagunit:actualResult>
        <tagunit:expectedResult><%= expectedResult %></tagunit:expectedResult>
</tagunit:assertEquals>
```

Summary of TagUnit Tags Part 1

Tag Name	Description
testTagLibrary ----- tagLibraryDescriptor	Specifies the tag library to be tested
assertNoAttributes	Asserts tag declares no attributes in TLD
assertAttribute	Asserts tag declares specified attribute in TLD
assertBodyContent	Asserts body content element in TLD has specified value
assertInterface	Asserts tag class implements specified interface
assertEquals ----- actualResult ----- expectedResult	Compares the output of a tag with the expected value
assertNotEquals ----- actualResult ----- expectedResult	Reverse of above
assertContains ----- actualResult ----- expectedResult	Checks that the output of a tag contains the expected value

Summary of TagUnit Tags Part 2

Tag Name	Description
assertMatches ----- actualResult ----- expectedResult	Checks output of a tag matches a regular expression
assertPageContextAttribute	Asserts an attribute can be found in the specified scope with the specified value or property
assertNoPageContextAttribute	Reverse of the above
assertException	Asserts that a tag throws an exception of the specified type
fail	Immediately fails a test (for use in unreachable code)
test	A general purpose grouping tag for assertion tags
runAs	Simulates the tag being run by the specified user and role
assertCustom ----- param	Lets you add new tests to TagUnit

Tag Files

- JSP 2.0 introduces a new kind of Tag Library
  - Designed to simplify the concept of custom actions
  - Especially for those who don't know Java well
- Sections of JSP code can be used as actions
  - The action is written using the JSP syntax
  - It is saved with the extension '.tag' or '.tagx'
- These segments are referred to as Tag Files
  - Extra directives and actions are available to support them
    - The new directives are 'tag', 'attribute' and 'variable'
    - The new actions are 'doBody' and 'invoke'
  - Like JSP's Tag Files can optionally use an XML syntax

Deploying Tag Files

- Tag Files can be deployed in two ways
  1. By placing them directly inside the Web App
  2. By placing them in a JAR inside ‘WEB-INF/lib’
- The only valid directory is ‘WEB-INF/tags’
  - Subdirectories are searched to any depth
- A Tag File in any other location is not valid
  - A JSP inside the Web App directory will be interpreted
  - A Tag File in that directory will be sent as is to the browser
    - In this sense a Tag File is not a ‘mini-JSP’

Deploying Tag Files

- The second way to deploy Tag Files is in a JAR
  - In which case they must be configured via a TLD
- A ‘tag-file’ element has been added to the TLD
  - This contains ‘name’ and ‘path’ elements
  - The path element is a file path relative to the root of the JAR
  - The file path must begin with ‘/META-INF/tags’
- This allows Tag Libraries to use both kinds of action
  - The library user doesn’t need to know the implementation
  - Two actions cannot have the same name even if one is a classic tag handler and the other is a Tag File

Including Tag Files

- Tag Files are included using the ‘taglib’ directive
  - This associates a prefix with a set of tags
- In JSP 2.0 this directive has a ‘tagdir’ attribute
  - This is used for Tag Files instead of the ‘uri’ attribute
- The value of ‘tagdir’ must be ‘/WEB-INF/tags’
  - Or a subdirectory inside ‘WEB-INF/tags’
- The container builds a set of implicit tag libraries
  - One for each directory under and including ‘/WEB-INF/tags’
  - You cannot use both ‘tagdir’ and ‘uri’

```
<%@ taglib prefix=“fmt” tagdir=“/WEB-INF/tags/formatting” %>
```

Implicit Objects in Tag Files

- Tag Files have the same implicit objects as JSP's
  - With the exception of the 'page' object
- Tag Files can include the output of other Tag Files
  - The include directive is used as in a JSP
  - The included file should have the extension '.tagf'

Implicit Reference	Type
request	HttpServletRequest
response	HttpServletResponse
pageContext	PageContext
session	HttpSession
application	ServletContext
out	JspWriter
config	ServletConfig

The Tag Directive

- This replaces the ‘page’ directive in Tag Files
  - It configures the way your Tag File will operate

Attribute	Description
display-name	Name used by IDE, defaults to tag name
body-content	The content of the tag, can be empty, tagdependant or scriptless
dynamic-attributes	Indicates support for dynamic attributes
small-icon and large-icon	Image files used by IDE, no default
description	A string that describes the tag, no default
example	A string that describes typical usage, no default
language	Same as page directive
import	Same as page directive
pageEncoding	Same as page directive
isELIgnored	Same as page directive

The Attribute Directive

- This allows you to specify attributes of your action
  - It does the same job as the attribute element in a TLD

Attribute	Description
name	The name of the attribute
required	Whether the attribute if required, default is true
fragment	Whether the attribute is a fragment to be evaluated, default is false (see the discussion of '<jsp:invoke>')
rtexprvalue	If the attribute value is a runtime expression, default is true
type	The type of the attribute value, must not be a primitive type NB cannot be used if the fragment attribute is set to true
description	The description of the attribute, no default

Example Tag File

```
<iter:iterator string="A#B#C#D#E" delim="#"/>
```

```
<%@ tag import="java.util.StringTokenizer,java.io.IOException"%>
<%@ attribute name="string" required="true" %>
<%@ attribute name="delim" required="true" %>
<ol>
    <%
        StringTokenizer st = new StringTokenizer(string,delim);
        try {
            while(st.hasMoreTokens()) {
                out.println("<li>" + st.nextToken() + "</li>");
            }
        } catch(IOException ex) {
            out.println("\tERROR " + ex);
        }
    %>
</ol>
```

The Variable Directive

- This allows you to add variables to the calling page
  - It does the same job as the variable element in a TLD

Attribute	Description
name-given	Defines the name to the variable to be created
name-from-attribute	Declares that the name of the variable will be the same as the given attribute
variable-class	The class name of the variable, default is string
scope	Scope of the variable, can be AT_BEGIN, AT_END or NESTED (the default)
declare	Whether the variable is declared after the tag invocation, defaults to true
description	The description of the variable, no default

The DoBody Action

- Using ‘doBody’ evaluates the body of the action
  - That is the content in the JSP page between the opening and closing tags of the Tag File
- By default the content is evaluated and written down the ‘JspWriter’ object to the browser
  - If the Tag has no content then nothing is written
- The attributes of ‘doBody’ can be used to redirect the evaluated content for further processing
  - Use ‘var’ or ‘varReader’ to name a String or Reader object in which you want the evaluated content stored
  - In either case the ‘scope’ attribute names one of the standard scopes where you want the variable added as an attribute

Example Tag File

```
<iter:iteratorV2 delim="#">A#B#C#D#E</iter:iteratorV2>
```

```
<%@ tag import="java.util.StringTokenizer,java.io.IOException"%>
<%@ attribute name="delim" required="true" %>
<jsp:doBody var="StringToBeTokenized" scope="page"/>
<ol>
    <%
        String string = (String)getJspContext().getAttribute("StringToBeTokenized",
                                                              PageContext.PAGE_SCOPE);
        StringTokenizer st = new StringTokenizer(string,delim);
        try {
            while(st.hasMoreTokens()) {
                out.println("<li>" + st.nextToken() + "</li>");
            }
        } catch(IOException ex) {  out.println("\tERROR " + ex); }
    %>
</ol>
```

The Invoke Action

- The ‘invoke’ action is similar to ‘doBody’
  - A fragment of content is evaluated and the result either written to ‘out’ or stored in a String or Reader
  - The ‘var’ , ‘varReader’ and ‘scope’ attributes have the same meaning as in the ‘doBody’ action
- However an arbitrary fragment is executed
  - The ‘fragment’ attribute names another Tag File attribute
  - The value of this attribute must be content to be evaluated
    - This second attribute must have been declared via an attribute directive with ‘fragment’ set to be true
    - The process of invoking a content fragment can be recursive