



## User Guide

[Table of Contents](#)

[Preface](#)

[Introduction](#)

[Model Components](#)

[View Components](#)

[Controller Components](#)

[Configuration](#)

[Release Notes](#)

[Installation](#)

## Developer Guides

[Bean Tags](#)

[HTML Tags](#)

[Logic Tags](#)

[Nested Tags](#)

[Template Tags](#)

[Tiles Tags](#)

[Utilities](#)

[Validator](#)

## Quick Links

[Welcome](#)

[News and Status](#)

[Resources](#)

[User and Developer Guides \\*](#)

[FAQs and HowTos](#)

## Table of Contents

- [0. Preface](#)
  - [0.1 The Usual Suspects](#)
  - [0.2 HTTP, HTML, and User Agents](#)
  - [0.3 The HTTP Request/Response Cycle](#)
  - [0.4 The Java Language and Application Frameworks](#)
  - [0.5 JavaBeans](#)
  - [0.6 Properties Files and ResourceBundle](#)
  - [0.7 Java Servlets](#)
  - [0.8 JavaServer Pages and JSP Tag Libraries](#)
  - [0.9 Extensible Markup Language](#)
  - [0.10 JAAS](#)
- [1. Introduction](#)
  - [1.1 Forward into the Past!](#)
  - [1.2 The Model-View-Controller \('MVC'\) Design Pattern](#)
    - [1.2.1 Struts Framework Overview](#)
    - [1.2.2 The Model: System State and Business Logic JavaBeans](#)
    - [1.2.3 The View: JSP Pages and Presentation Components](#)
  - [1.3 The Controller: ActionServlet and ActionMapping](#)
- [2. Building Model Components](#)
  - [2.1 Overview](#)
  - [2.2 JavaBeans and Scope](#)
  - [2.3 ActionForm Beans](#)
  - [2.4 System State Beans](#)
  - [2.5 Business Logic Beans](#)
  - [2.6 Accessing Relational Databases](#)
- [3. Building View Components](#)
  - [3.1 Overview](#)
  - [3.2 Internationalization](#)
  - [3.3 Forms and FormBean Interactions](#)
    - [3.3.1 Indexed & Mapped Properties](#)
    - [3.3.2 Input Field Types Supported](#)
    - [3.3.3 Other Useful Presentation Tags](#)
    - [3.3.4 Automatic Form Validation](#)
    - [3.3.5 Struts Validator](#)
  - [3.4 Other Presentation Techniques](#)
    - [3.4.1 Application-Specific Custom Tags](#)
    - [3.4.2 Page Composition With Includes](#)
    - [3.4.3 Page Composition With Tiles](#)
    - [3.4.4 Image Rendering Components](#)
    - [3.4.5 Rendering Text](#)
    - [3.4.6 The Struts-EL Tag Library](#)
- [4. Building Controller Components](#)

- [4.1 Overview](#)
  - [4.2 The ActionServlet](#)
    - [4.2.1 Request Processor](#)
  - [4.3 ActionForm Classes](#)
    - [4.3.1 DynaActionForm Classes](#)
    - [4.3.2 Map-backed ActionForm Classes](#)
  - [4.4 Action Classes](#)
    - [4.4.1 Action Class Design Guidelines](#)
  - [4.5 Exception Handler](#)
  - [4.6 Plugin Classes](#)
  - [4.7 The ActionMapping Implementation](#)
  - [4.8 Writing ActionMappings](#)
    - [4.8.1 ActionMapping Example](#)
  - [4.9 Using ActionMappings for Pages](#)
  - [4.10 Using The Commons Logging Interface](#)
- 5. Configuring Applications
  - [5.1 Overview](#)
  - [5.2 The Struts Configuration File](#)
    - [5.2.1 Controller Configuration](#)
    - [5.2.2 Message Resources Configuration](#)
    - [5.2.3 PlugIn Configuration](#)
    - [5.2.4 Data Source Configuration](#)
    - [5.3 Configuring your application for modules](#)
    - [5.3.3 Switching Modules](#)
    - [5.4 The Web Application Deployment Descriptor](#)
    - [5.5 Add Struts Components To Your Application](#)
  - [5.3 The Web Application Deployment Descriptor](#)
    - [5.3.1 Configure the Action Servlet Instance](#)
    - [5.3.2 Configure the Action Servlet Mapping](#)
    - [5.3.3 Configure the Struts Tag Library](#)
      - [5.3.3.1 Configure the Struts Tag Library \(Servlet 2.3\)](#)
  - [5.4 Add Struts Components To Your Application](#)
- 6. Getting Started
  - [6.1 Release Notes](#)
    - [Release Notes 1.1-b3](#)
    - [Release Notes 1.1-b2](#)
    - [Release Notes 1.1-b1](#)
    - [Release Notes 1.0.2](#)
    - [Release Notes 1.0.1](#)
    - [Release Notes 1.0](#)
    - [Release Notes 1.0-b3](#)
    - [Release Notes 1.0-b2](#)
    - [Release Notes 1.0-b1](#)
  - [6.2 Installation](#)
    - [iPlanet](#)
    - [Portal Application Server](#)
    - [Jetty](#)
    - [JRun 3.0](#)
    - [Orion Application Server](#)
    - [SilverStream Application Server 3.7.1 and later](#)
    - [Tomcat with Apache](#)

- [Bluestone Universal Business Server 7.2](#)
- [WebSphere Application Server 3.5 FixPack 2](#)
- [WAS with the Example Application](#)
- [Weblogic 5.1 sp8](#)

Next: [Preface](#)

---

*Copyright (c) 2000-2002, Apache Software Foundation*

**Powered by**  
**Struts**



## User Guide

[Table of Contents](#)[Preface](#)[Introduction](#)[Model](#)[Components](#)[View](#)[Components](#)[Controller](#)[Components](#)[Configuration](#)[Release Notes](#)[Installation](#)

## Developer Guides

[Bean Tags](#)[HTML Tags](#)[Logic Tags](#)[Nested Tags](#)[Template Tags](#)[Tiles Tags](#)[Utilities](#)[Validator](#)

## Quick Links

[Welcome](#)[News and Status](#)[Resources](#)[User and  
Developer  
Guides \\*](#)[FAQs and  
HowTos](#)

## 0. Preface: Core Technologies

Contributors:

- Ted Husted
- Ed Burns
- Craig R. McClanahan

### 0.1 The Usual Suspects

This User Guide is written for active web developers and assumes a working knowledge about how Java web applications are built. Before getting started, you should understand the basics of several core technologies:

- [HTTP, HTML, and User Agents](#)
- [The HTTP Request/Response Cycle](#)
- [The Java Language and Application Frameworks](#)
- [JavaBeans](#)
- [Properties Files and ResourceBundles](#)
- [Java Servlets](#)
- [JavaServer Pages and JSP Tag Libraries](#)
- [Extensible Markup Language](#)

This chapter briefly defines each of these technologies but does not describe them in detail. For your convenience, links to further information are provided if you would like to learn more about a technology.

If you are familiar with Java, but not these technologies, the best overall starting point is [The Java Web Services Tutorial](#). This is also available for download in [PDF](#) format.

If you've created web applications for other platforms, you may be able to follow along and visit the other references as needed. The core technologies used by Struts are also used by most other Java web development products, so the background information will be useful in any Java project.

If you are not familiar with the Java language generally, then the best starting point is [The Java Tutorial](#). This overlaps with the Java Web Services Tutorial in some places, but the two work well together.

### 0.2 HTTP, HTML and User Agents

The World Wide Web was built over the Hypertext Transfer Protocol ([HTTP](#)) and the Hypertext Markup Language ([HTML](#)). A User Agent, like a web browser, uses HTTP to request a HTML document. The browser then formats and displays the document to its user. HTTP is used to transport more than HTML, but HTML is the lingua franca of the Web and web applications.

While building web applications, some Java developers will write their own HTML. Others leave that responsibility to the page designers.

For more about HTTP, HTML, and User Agents, see:

- [Getting started with HTML](#) by Dave Raggett
- [HTTP Overview](#) in the Java Web Services Tutorial.
- [HTTP/1.1](#) Specification
- [HTTP Basic and Digest Authentication](#) Specification
- [State Management Mechanism](#) Specification (Cookies)

## 0.3 The HTTP Request/Response cycle

A very important part of HTTP for the web developer is the request/response cycle. To use HTTP you have to make a request. A HTTP server, like a web server, is then obliged to respond. When you build your web application, you design it to react to a HTTP request by returning a HTTP response. Frameworks like Struts abstract much of these nuts and bolts, but it is important to understand what is happening behind the scenes.

If you are not familiar with the HTTP request/response cycle, we **strongly** recommend the [HTTP Overview](#) in the Java Web Services Tutorial.

## 0.4 The Java Language and Application Frameworks

Struts is written in the popular and versatile [Java programming language](#). Java is an object-orientated language, and Struts makes good use of many object-orientated techniques. In addition, Java natively supports the concept of *threads*, which allows more than one task to be performed at the same time. A good understanding of Java, and especially object-orientated programming (OOP) and threading, will help you get the most out of Struts and this User Guide.

For more about Java and threads, see

- [Learning the Java Language](#) in the Java Tutorial
- [Threads: Doing Two or More Tasks At Once](#) in the Java Language Tutorial

Even if you have worked with Java and OOP before, it can also help to be aware of the programming challenges specific to creating and using application frameworks. For more about application frameworks, see the classic white papers

- [Designing Reusable Classes](#) by Ralph E. Johnson & Brian Foote
- [Object-Oriented Application Frameworks](#) by Mohamed Fayad and Douglas C. Schmidt

These papers can be especially helpful if you are [fact-finding or reviewing](#) server-side [frameworks](#).

## 0.5 JavaBeans

Like many Java applications, most of the Struts objects are designed as [JavaBeans](#). Following the JavaBean design patterns makes the Struts classes easier to use -- both by Java developers and by Java development tools.

Although JavaBeans were first created for visual elements, these object design patterns have been found to be useful as the basis for any reusable component, like those used by the Struts framework.

For more about JavaBeans, see:

- [The JavaBeans Component Architecture Documentation](#) page at `java.sun.com`, including a link to download the [JavaBeans 1.01 Specification](#)
- [The JavaBean Trail](#) in the Java Tutorial
- [JavaBeans Components in JSP Pages](#) in the Java Web Services Tutorial

### 0.5.1 Reflection and Introspection

Reflection is the process of determining which member fields and methods are available on an object. Introspection is a specialized form of reflection used by the JavaBean API. Using Introspection, we can determine which methods of a JavaBean are intended to be accessed by other objects. (The getters and the setters, for example.)

The Struts framework uses Introspection to convert HTTP parameters into JavaBean properties and to populate HTML fields from JavaBean properties. This technique makes it easy to "roundtrip" properties between HTML forms and JavaBeans.

For more about Reflection and Introspection, see

- [The Reflection Trail](#)
- [Chapter 8 of the JavaBeans API Specification](#)

### 0.5.2 Maps

JavaBeans store data as properties and may act on that data through other methods. JavaBeans are flexible and powerful objects but are not the only object that programmers use to store data. Another popular object is the Map [`java.util.Map`]. A Map is a simple collection of name and value pairs. Maps are often used "behind the scenes" as a flexible way to store dynamic data.

### 0.5.3 DynaBeans

DynaBeans combine the extensibility of JavaBeans with the flexibility of a Map. Defining even the simplest JavaBean requires defining a new class and coding a field and two methods for each property. The properties of a DynaBean can be configured via an XML descriptor. The virtual properties of a DynaBean can't be called by standard Java methods, but work well with components that rely on reflection and introspection.

In a Struts application, you can use DynaBeans to describe your HTML forms. This strategy can avoid creating a formal JavaBean subclass to store a few simple properties.

For more about DynaBeans, see

- [The Commons BeanUtils Javadocs](#)

## 0.6 Properties Files and ResourceBundles

Java applications, including web applications, are often configured using [Properties](#) files. Properties files are the basis for the [ResourceBundles](#) that Struts uses to provide message resources to an application.

For more about Properties files, see:

- [Using Properties to Manage Program Attributes](#) in The Java Tutorial

Java ResourceBundles use one or more Properties files to provide internationalized messages to users based their [Locale](#). Support for localizing an application was built into Struts from the ground-up.

For more about localization and ResourceBundles, see

- [About the ResourceBundle Class](#) in the Java Tutorial

## 0.7 Java Servlets

Since Java is an object-orientated language, the [Java Servlet](#) platform strives to cast HTTP into an object-orientated form. This strategy makes it easier for Java developers to concentrate on what they need their application to do -- rather than the mechanics of HTTP.

HTTP provides a standard mechanism for extending servers called the Common Gateway Interface, or CGI. The server can pass a request to a CGI-aware program, and the program will pass back a response. Likewise, a Java-aware server can pass a request to a servlet container. The container can fulfill the request or it can pass the request back to the HTTP server. The container decides whether it can handle the request by checking its list of servlets. If there is a servlet registered for the request, the container passes the request to the servlet.

When a request comes in, the container checks to see if there is a servlet registered for that request. If there is a match, the request is given to the servlet. If not, the request is returned to the HTTP server.

It's the container's job to manages the servlet lifecycle. The container creates the servlets, invokes the servlets, and ultimately disposes the servlets.

A servlet is generally a subclass of `javax.servlet.http.HttpServlet`. A servlet must implement four methods, which are invoked by the container as needed:

- **public void init(ServletConfig config)** - Called by the servlet container when the servlet instance is first created, and before any request is processed.
- **public void doGet(HttpServletRequest request, HttpServletResponse response)** - Called to process a specific request received using the HTTP GET protocol, which generates a corresponding dynamic response.
- **public void doPost(HttpServletRequest request, HttpServletResponse response)** - Called to process a specific request received using the HTTP POST protocol, which generates a corresponding dynamic response.
- **public void destroy()** - Called by the servlet container when it takes this servlet instance out of service, such as when a web application is being undeployed or when the entire container is being shut down.

Struts provides a ready-to-use servlet for your application [`org.apache.struts.action.ActionServlet`]. As a Struts developer, you can then just write objects that the Struts ActionServlet calls when needed. But it is still helpful to understand the basics of what servlets are, and the role they play in a Java web application.

For more about Java Servlets, see:

- [The Java Servlet Technology](http://java.sun.com/javase/6/docs/api/javax/servlet/) page at `java.sun.com`
- [The Servlet 2.2 and 2.3 Specifications](http://java.sun.com/javase/6/docs/api/javax/servlet/) download page at `java.sun.com`
- [Java Servlet Technology](http://java.sun.com/javase/6/docs/api/javax/servlet/) in the Java Web Services Tutorial.
- [Web Applications](http://java.sun.com/javase/6/docs/api/javax/servlet/) in the Java Web Services Tutorial.

## 0.7.1 Servlets and threads

To boost performance, the container can multi-thread servlets. Only one instance of a particular servlet is created, and each request for that servlet passes through the same object. This strategy helps the container make the best use of available resources. The tradeoff is that the servlet's `doGet()` and `doPost()` methods must be programmed in a *thread-safe* manner.

For more about servlets and thread-safety, see:

- [Controlling Concurrent Access to Shared Resources](http://java.sun.com/javase/6/docs/api/javax/servlet/) in Java Web Services Tutorial.

## 0.7.2 Servlet Context

The *ServletContext* interface [`javax.servlet.ServletContext`] defines a servlet's view of the web application within which the servlet is running. It is accessible in a servlet via the `getServletConfig()` method, and in a JSP page as the application implicit variable. Servlet contexts provide several APIs that are very useful in building Struts based web applications:

- *Access To Web Application Resources* - A servlet can access static resource files within the web application using the `getResource()` and `getResourceAsStream()` methods.
- *Servlet Context Attributes* - The context makes available a storage place for Java objects, identified by string-valued keys. These attributes are global to the entire web application, and may be accessed by a servlet using the `getAttribute()`,



`getAttributeNames()`, `removeAttribute()`, and `setAttribute()` methods. From a JSP page, servlet context attributes are also known as "application scope beans".

For more about the servlet context, see:

- [Accessing the Web Context](#) in Java Web Services Tutorial.

### 0.7.3 Servlet Request

Each request processed by a servlet is represented by a Java interface, normally a `HttpServletRequest` [`javax.servlet.http.HttpServletRequest`]. The request interface provides an object-oriented mechanism to access all of the information that was included in the underlying HTTP request, including:

- *Cookies* - The set of cookies included with this request are available via the `getCookies()` method.
- *Headers* - HTTP headers that were included with the request are accessible by name. You can enumerate the names of all included headers.
- *Parameters* - Request parameters, including those from the query string portion of the URL and from the embedded content of the request (POST only) are available by name.
- *Request Characteristics* - Many other characteristics of the incoming HTTP request, such as the method used (normally GET or POST) the protocol scheme used ("http" or "https"), and similar values.
- *Request URI Information* - The original request URI being processed is available via `getRequestURI()`. In addition, the constituent parts into which the servlet container parses the request URI (`contextPath`, `servletPath`, and `pathInfo`) are available separately.
- *User Information* - If you are using [Container Managed Security](#), you can ask for the username of the authenticated user, retrieve a `Principal` object representing the current user, and whether the current user is authorized for a specified role.

In addition, servlet requests support *request attributes* (from JSP, these are "request scope beans"), analogous to the servlet context attributes described above. Request attributes are often used to communicate state information from a business logic class that generates it to a view component (such as a JSP page) that will use the information to produce the corresponding response.

The servlet container guarantees that a particular request will be processed by a servlet on a single thread. Therefore, you do not generally have to worry about the thread safety of your access to request properties and attributes.

For more about the servlet request, see:

- [Getting Information from Requests](#) in Java Web Services Tutorial.

### 0.7.4 Servlet Response

The primary purpose of a servlet is to process an incoming [Servlet Request](#) [`javax.servlet.http.HttpServletRequest`] and convert it into a corresponding response. This is performed by calling appropriate methods on the servlet response [`javax.servlet.http.HttpServletResponse`] interface. Available methods let you:

- *Set Headers* - You can set HTTP headers that will be included in the response. The most important header is the `Content-Type` header, which tells your client what kind of information is included in the body of this response. This is typically set to `text/html` for an HTML page, or `text/xml` for an XML document.
- *Set Cookies* - You can add cookies to the current response.
- *Send Error Responses* - You can send an HTTP error status (instead of a usual page of content) using `sendError()`.
- *Redirect To Another Resource* - You can use the `sendRedirect()` method to redirect the client to some other URL that you specify.

An important principle in using the servlet response APIs is that any methods you call to manipulate headers or cookies **MUST** be performed before the first buffer-full of content has been flushed to the client. The reason for this restriction is that such information is transmitted at the beginning of the HTTP response, so trying things like adding a header after the headers have already been sent will not be effective.

When you are using presentation pages in a Model 2 application, you will not generally use the servlet response APIs directly. In the case of `JavaServerPages`, the JSP page compiler in your servlet container will convert your page into a servlet. The JSP servlet renders the response, interspersing dynamic information where you have interposed JSP custom tags.

Other presentation systems, like Velocity Tools for Struts, may delegate rendering the response to a specialized servlet, but the same pattern holds true. You create a template, and the dynamic response is generated automatically from the template.

For more about the servlet response, see:

- [Constructing Responses](#) in Java Web Services Tutorial.

## 0.7.5 Filtering

If you are using a servlet container based on version **2.3** or later of the Servlet Specification (such as Tomcat 4.x), you can take advantage of the new Filter APIs [`javax.servlet.Filter`] that let you compose a set of components that will process a request or response. Filters are aggregated into a chain in which each filter has a chance to process the request and response before and after it is processed by subsequent filters (and the servlet that is ultimately called).

Struts 1.0 and 1.1 require only version 2.2 or later of the Servlet Specification to be implemented by your servlet container, so Struts does not itself utilize Filters at this time. However, you can still use them yourself if you are running on a 2.3 or later container. It is very likely that future versions of Struts will require a Servlet 2.3 or later container, so that Struts itself can utilize filters.

For more about filters, see:

- [Filtering Requests and Responses](#)

## 0.7.6 Sessions

One of the key characteristics of HTTP is that it is *stateless*. In other words, there is nothing built in to HTTP that identifies a subsequent request from the same user as being related to a previous request from that user. This makes building an application that wants to engage in a conversation with the user over several requests to be somewhat difficult.

To alleviate this difficulty, the servlet API provides a programmatic concept called a *session*, represented as an object that implements the `javax.servlet.http.HttpSession` interface. The servlet container will use one of two techniques (cookies or URL rewriting) to ensure that the next request from the same user will include the *session id* for this session, so that state information saved in the session can be associated with multiple requests. This state information is stored in *session attributes* (in JSP, they are known as "session scope beans").

To avoid occupying resources forever when a user fails to complete an interaction, sessions have a configurable *timeout interval*. If the time gap between two requests exceeds this interval, the session will be timed out, and all session attributes removed. You define a default session timeout in your web application deployment descriptor, and you can dynamically change it for a particular session by calling the `setMaxInactiveInterval()` method.

Unlike requests, you need to be concerned about thread safety on your session attributes (the methods these beans provide, not the `getAttribute()` and `setAttribute()` methods of the session itself). It is surprisingly easy for there to be multiple simultaneous requests from the same user, which will therefore access the same session.

Another important consideration is that session attributes occupy memory in your server *in between* requests. This can have an impact on the number of simultaneous users that your application can support. If your application requirements include very large numbers of simultaneous users, you will likely want to minimize your use of session attributes, in an effort to control the overall amount of memory required to support your application.

For more about sessions, see:

- [Maintaining Client State](#) in Java Web Services Tutorial
- [javax.servlet.http.HttpSession](#)

## 0.7.7 Dispatching Requests

The Java Servlet specification extends the HTTP request/response cycle by allowing the request to be dispatched, or forwarded, between resources. Struts uses this feature to pass a request through specialized components, each handling one aspect of the response. In the normal course, a request may pass through a controller object, a model object, and finally to a view object as part of a single request/response cycle.

## 0.7.8 Web Applications

Just as a HTTP server can be used to host several distinct web sites, a servlet container can be used to host more than one web application. The Java servlet platform provides a well-defined mechanism for organizing and deploying web applications. Each application runs in its own namespace so that they can be developed and deployed separately. A web application can be assembled into a Web Application Archive, or WAR file. The single WAR can be uploaded to the server and automatically deployed.

For more about web applications, see:

- [Web Applications](#) in Java Web Services Tutorial

## 0.7.9 Web application deployment descriptor (web.xml)

Most aspects of an application's lifecycle are configured through an XML document called the Web application deployment descriptor. The schema of the descriptor, or web.xml, is given by the Java servlet specification.

For more about the web.xml and application lifecycle events, see:

- [Web Application Life Cycle](#) in Java Web Services Tutorial

## 0.7.10 Security

One detail that can be configured in the Web application deployment descriptor is container-managed security. Declarative security can be used to protect requests for URIs that match given patterns. Pragmatic security can be used to fine-tune security make authorization decisions based on the time of day, the parameters of a call, or the internal state of a Web component. It can also be used to restrict authentication based on information in a database.

For more information about container-managed security, see:

- [Web Application Security](#) in the Java Web Services Tutorial

## 0.8 JavaServer Pages and JSP Tag Libraries

[JavaServer Pages](#) (JSPs) are "inside-out servlets" that make it easier to create and maintain dynamic web pages. Instead of putting what you want to write to the HTTP response inside of a Java `print` statement, everything in a JavaServer Page is written to the response, **except** what is placed within special Java statements.

With JavaServer Pages you can start by writing the page in standard HTML and then add the dynamic features using statements in the Java language or by using [JSP tags](#). The Struts distribution includes several JSP tags that make it easy to access the framework's features from a JavaServer Page.

For more about JavaServerPages and JSP Tag Libraries see

- [The JavaServer Pages Technology](#) page at `java.sun.com`
- [The JSP 1.1 and 1.2 Specifications](#) download page at `java.sun.com`
- [JavaServer Pages Technology](#) in the Java Web Services Tutorial

- [Custom Tags in JSP Pages](#) in the Java Web Services Tutorial

Many times, JSP tags work hand-in-hand with JavaBeans. The application sends a JavaBean to the JSP, and the JSP tag uses the bean to customize the page for the instant user. For more, see [JavaBeans Components in JSP Pages](#) in the Java Web Services Tutorial.

Struts also works well with the new [JavaServer Pages Standard Tag Library](#) and taglibs from other sources, like [JSP Tags](#) and [Jakarta Taglibs](#).

One of the contributed libraries available for Struts is **Struts-EL**. This taglib is specifically designed to work well with the JavaServer Pages Standard Tag Library. In particular, it uses the same "expression language" engine for evaluating tag attribute values as the JSTL. This is in contrast to the original Struts tag library, which can only use "rtexprvalue"s (runtime scriptlet expressions) for dynamic attribute values.

There are also toolkits available that make Struts easy to use with [XSLT](#) and [Velocity Templates](#).

## 0.9 Extensible Markup Language (XML)

The features provided by the Struts framework relies on a number of objects that are usually deployed using a configuration file written in [Extensible Markup Language](#). XML is also used to configure Java web applications; so, this is yet another familiar approach.

For more about XML configuration files and Java web applications, see

- [Configuring Web Applications](#) in the Java Web Services Tutorial

For more about how XML is used with Java applications generally, see [Java API for XML Processing](#) in the Java Web Services Tutorial. While the framework makes good use of this API internally, it is not something most Struts developers would use when writing their own applications.

### 0.9.1 Descriptors

When Java applications use XML configuration files, the elements are most often used as *descriptors*. The application does not use the XML elements directly. The elements are used to create and configure (or deploy) Java objects.

The Java Servlet platform uses an XML configuration file to deploy servlets (among other things). Likewise, Struts uses an XML configuration file to deploy objects used by the framework.

## 0.10 Other layers

Struts provides the control layer for a web application. Developers can use this layer with other standard technologies to provide the data access and presentation layers. Some popular Data access technologies include:

- [Castor](#)
- [Enterprise Java Beans](#)
- [JDBC](#)
- [Object Relational Bridge](#)
- [Simper](#)

Presentation layer technologies include:

- [JavaServer Pages](#)
- [Velocity Templates](#)
- [XSL Transformations](#)

Other [data access](#) and [presentation](#) systems are listed in the Struts [Resource Guide](#).

## 0.11 JAAS

While Struts can work with any approach to user authentication and authorization, Struts 1.1 will offer direct support for the standard Java Authentication and Authorization Service (JAAS). In Struts 1.1 (and the current nightly build), you can specify security roles on an action-by-action basis.

For more about JAAS, see the [Javasoft product page](#) and the [Web Application Security](#) chapter of the [Java Web Services Tutorial](#).

Next: [Introduction](#)



## User Guide

[Table of Contents](#)[Preface](#)[Introduction](#)[Model](#)[Components](#)[View](#)[Components](#)[Controller](#)[Components](#)[Configuration](#)[Release Notes](#)[Installation](#)

## Developer Guides

[Bean Tags](#)[HTML Tags](#)[Logic Tags](#)[Nested Tags](#)[Template Tags](#)[Tiles Tags](#)[Utilities](#)[Validator](#)

## Quick Links

[Welcome](#)[News and Status](#)[Resources](#)[User and  
Developer  
Guides \\*](#)[FAQs and  
HowTos](#)

## 1. Introduction

Contributors:

- Craig R. McClanahan
- Mike Schachter
- Larry McCay
- Ted Husted
- Martin Cooper
- Ed Burns
- Dominique Plante

### 1.1 Forward into the Past! (or a brief history of Struts)

When Java servlets were first invented, many programmers quickly realized that they were a Good Thing. They were faster and more powerful than standard CGI, portable, and infinitely extensible.

But writing HTML to send to the browser in endless `println()` statements was tiresome and problematic. The answer to that was [JavaServer Pages](#), which turned [Servlet](#) writing inside-out. Now developers could easily mix HTML with Java code, and have all the advantages of servlets. The sky was the limit!

Java web applications quickly became "JSP-centric". This in-and-of itself was not a Bad Thing, but it did little to resolve flow control issues and other problems endemic to web applications.

Another model was clearly needed ...

Many clever developers realized that JavaServer Pages AND servlets could be used **together** to deploy web applications. The servlets could help with the control-flow, and the JSPs could focus on the nasty business of writing HTML. In due course, using JSPs and servlets together became known as [Model 2](#) (meaning, presumably, that using JSPs alone was Model 1).

Of course, there is nothing new under the Sun ... and many have been quick to point out that JSP's Model 2 follows the classic [Model-View-Controller](#) design pattern abstracted from the venerable [Smalltalk MVC framework](#). Java Web developers now tend to use the terms Model 2 and MVC interchangeably. In this guide, we use the MVC paradigm to describe the Struts architecture, which might be best termed a Model 2/MVC design.

The Struts project was launched in May 2000 by Craig R. McClanahan to provide a standard MVC framework to the Java community. In July 2001, Struts 1.0 was released, and IOHO, Java Model 2 development will never be quite the same.

### 1.2 The Model-View-Controller ('MVC') Design Pattern



In the MVC design pattern, application flow is mediated by a central Controller. The Controller delegates requests - in our case, HTTP requests - to an appropriate handler. The handlers are tied to a Model, and each handler acts as an adapter between the request and the Model. The Model represents, or encapsulates, an application's business logic or state. Control is usually then forwarded back through the Controller to the appropriate View. The forwarding can be determined by consulting a set of mappings, usually loaded from a database or configuration file. This provides a loose coupling between the View and Model, which can make applications significantly easier to create and maintain.

## 1.2.1 The Model: System State and Business Logic JavaBeans

The *Model* portion of an MVC-based system can be often be divided into two major subsystems -- the **internal state** of the system and the **actions** that can be taken to change that state.

In grammatical terms, we might think about state information as **nouns** (things) and actions as **verbs** (changes to the state of those things).

Many applications represent the internal state of the system as a set of one or more JavaBeans. The bean properties represent the details of the system's state. Depending on your application's complexity, these beans may be self contained (and know how to persist their own state), or they may be facades that know how to retrieve the system's state from another component. This component may be a database, a search engine, an Entity Enterprise JavaBean, a LDAP server, or something else entirely.

Large-scale applications will often represent the set of possible business operations as methods that can be called on the bean or beans maintaining the state information. For example, you might have a shopping cart bean, stored in session scope for each current user, with properties that represent the current set of items that the user has decided to purchase. This bean might also have a `checkout()` method that authorizes the user's credit card and sends the order to the warehouse to be picked and shipped. Other systems will represent the available operations separately, perhaps as Session Enterprise JavaBeans (Session EJBs).

In a smaller scale application, on the other hand, the available operations might be embedded within the `Action` classes that are part of the Struts control layer. This can be useful when the logic is very simple or where reuse of the business logic in other environments is not contemplated.

The Struts framework architecture is flexible enough to support most any approach to accessing the Model, but we **strongly** recommend that you separate the business logic ("how it's done") from the role that `Action` classes play ("what to do"). 'nuff said.

For more about adapting your application's Model to Struts, see the [Building Model Components](#) chapter.

## 1.2.2 The View: JSP Pages and Presentation Components



The *View* portion of a Struts-based application is most often constructed using JavaServer Pages (JSP) technology. JSP pages can contain static HTML (or XML) text called "template text", plus the ability to insert dynamic content based on the interpretation (at page request time) of special action tags. The JSP environment includes a set of standard action tags, such as `<jsp:useBean>` whose purpose is described in the [JavaServer Pages Specification](#). In addition to the built-in actions, there is a standard facility to define your own tags, which are organized into "custom tag libraries."

Struts includes a set of custom tag libraries that facilitate creating user interfaces that are fully internationalized, and that interact gracefully with `ActionForm` beans that are part of the *Model* portion of the system.

For more about the Struts taglibs and using presentation pages with the framework, see the "[Building View Components](#)" chapter. Additional documentation regarding the taglibs is also available in the Developer Guides (see menu).

### 1.2.3 The Controller: `ActionServlet` and `ActionMapping`

The *Controller* portion of the application is focused on receiving requests from the client (typically a user running a web browser), deciding what business logic function is to be performed, and then delegating responsibility for producing the next phase of the user interface to an appropriate View component. In Struts, the primary component of the Controller is a servlet of class `ActionServlet`. This servlet is configured by defining a set of `ActionMappings`. An `ActionMapping` defines a path that is matched against the request URI of the incoming request, and usually specifies the fully qualified class name of an Action class. All Actions are subclassed from `org.apache.struts.action.Action`. Actions encapsulate the business logic, interpret the outcome, and ultimately dispatch control to the appropriate View component to create the response.

Struts also supports the ability to use `ActionMapping` classes that have additional properties beyond the standard ones required to operate the framework. This allows you to store additional information specific to your application, but still utilize the remaining features of the framework. In addition, Struts lets you define logical "names" to which control should be forwarded so that an action method can ask for the "Main Menu" page (for example), without knowing what the actual name of the corresponding JSP page is. These features greatly assist you in separating the control logic (what to do) with the view logic (how it's rendered).

For more about the Struts control layer, see the [Building Controller Components](#) chapter.

## 1.3 Struts Control Flow

The Struts framework provides several components that make up the **Control** layer of a MVC-style application. These include a controller servlet, developer-defined request handlers, and several supporting objects.

The Struts custom tag libraries provide direct support for the **View** layer of a MVC application. Some of these access the control-layer objects. Others are generic tags found convenient when writing applications. Other taglibs, including [JSTL](#), can also be used with Struts. Other presentation technologies, like [Velocity Templates](#) and [XSLT](#) can also be used with Struts.

The **Model** layer in a MVC application is often project-specific. Struts is designed to makes it easy to access the business-end of your application, but leaves that part of the programming to other products, like [JDBC](#), [Enterprise Java Beans](#), [Object Relational Bridge](#), or [Simper](#), to name a few.

Let's step through how this all fits together.

When initialized, the controller parses a configuration file (`struts-config.xml`) and uses it to deploy other control layer objects. Together, these objects form the **Struts Configuration**. The Struts Configuration defines (among other things) the [ActionMappings](#) [`org.apache.struts.action.ActionMappings`] for an application.

The Struts controller servlet consults the ActionMappings as it routes HTTP requests to other components in the framework. Requests may be forwarded to JavaServer Pages or [Action](#) [`org.apache.struts.action.Action`] subclasses provided by the Struts developer. Often, a request is first forwarded to an Action and then to a JSP (or other presentation page). The mappings help the controller turn HTTP requests into application actions.

An individual [ActionMapping](#) [`org.apache.struts.action.ActionMapping`] will usually contain a number of properties including:

- a **request path** (or "URI"),
- the **object type** (Action subclass) to act upon the request, and
- other properties as needed.

The Action object can handle the request and respond to the client (usually a Web browser) or indicate that control should be forwarded elsewhere. For example, if a login succeeds, a login action may wish to forward the request onto the mainMenu page.

Action objects have access to the application's controller servlet, and so have access to that servlet's methods. When forwarding control, an Action object can indirectly forward one or more shared objects, including [JavaBeans](#), by placing them in one of the standard contexts shared by Java Servlets.

For example, an Action object can create a shopping cart bean, add an item to the cart, place the bean in the session context, and then forward control to another mapping. That mapping may use a JavaServer Page to display the contents of the user's cart. Since each client has their own session, they will each also have their own shopping cart.

In a Struts application, most of the business logic can be represented using JavaBeans. An Action can call the properties of a JavaBean without knowing how it actually works. This encapsulates the business logic, so that the Action can focus on error handling and where to forward control.

JavaBeans can also be used to manage input forms. A key problem in designing Web

applications is retaining and validating what a user has entered between requests. With Struts, you can define your own set of input bean classes, by subclassing [ActionForm](#) [`org.apache.struts.action.ActionForm`]. The ActionForm class makes it easy to store **and validate** the data for your application's input forms. The ActionForm bean is automatically saved in one of the standard, shared context collections, so that it can be used by other objects, like an Action object or another JSP.

The form bean can be used by a JSP to collect data from the user ... by an Action object to validate the user-entered data ... and then by the JSP again to re-populate the form fields. In the case of validation errors, Struts has a shared mechanism for raising and displaying error messages.

Another element of the Struts Configuration are the [ActionFormBeans](#) [`org.apache.struts.action.ActionFormBeans`]. This is a collection of [descriptor objects](#) that are used to create instances of the ActionForm objects at runtime. When a mapping needs an ActionForm, the servlet looks up the form-bean descriptor by name and uses it to create an ActionForm instance of the specified type.

Here is the sequence of events that occur when a request calls for an mapping that uses an ActionForm:

- The controller servlet either retrieves or creates the ActionForm bean instance.
- The controller servlet passes the bean to the Action object.
- If the request is being used to submit an input page, the Action object can examine the data. If necessary, the data can be sent back to the input form along with a list of messages to display on the page. Otherwise the data can be passed along to the business tier.
- If the request is being used to create an input page, the Action object can populate the bean with any data that the input page might need.

The Struts framework includes custom tags that can automatically populate fields from a JavaBean. All most JavaServer Pages really need to know about the rest of the framework is the field names to use and where to submit the form.

Other Struts tags can automatically output messages queued by an Action or ActionForm and simply need to be integrated into the page's markup. The messages are designed for [localization](#) and will render the best available message for a user's locale.

The Struts framework and its custom tag libraries were designed from the ground-up to support the internationalization features built into the Java platform. All the field labels and messages can be retrieved from a [message resource](#). To provide messages for another language, simply add another file to the resource bundle.

Internationalism aside, other benefits to the message resources approach are consistent labeling between forms, and the ability to review all labels and messages from a central location.

For the simplest applications, an Action object may sometimes handle the business logic associated with a request. **However, in most cases, an Action object should invoke another object, usually a JavaBean, to perform the actual business logic.** This lets the Action focus on error handling and control flow, rather than business logic. To allow reuse on other platforms, business-logic JavaBeans should not refer to any Web application objects. The Action object should translate needed details from the HTTP request and pass those along to the business-logic beans as regular Java variables.

In a database application, for example:

- A business-logic bean will connect to and query the database,
- The business-logic bean returns the result to the Action,
- The Action stores the result in a form bean in the request,
- The JavaServer Page displays the result in a HTML form.

Neither the Action nor the JSP need to know (or care) from where the result comes. They just need to know how to package and display it.

[Other chapters](#) in this document cover the various Struts components in greater detail. The Struts release also includes several Developer Guides covering various aspects of the frameworks, along with sample applications, the standard Javadoc API, and, of course, the complete source code!

Struts is distributed under the Apache Software Foundation license. The code is copyrighted, but is free to use in any application. See the [ASF license](#) for specifics.

Next: [Building Model Components](#)

---

*Copyright (c) 2000-2002, Apache Software Foundation*

**Powered by  
Struts**



## User Guide

[Table of Contents](#)

[Preface](#)

[Introduction](#)

[Model](#)

[Components](#)

[View](#)

[Components](#)

[Controller](#)

[Components](#)

[Configuration](#)

[Release Notes](#)

[Installation](#)

## Developer Guides

[Bean Tags](#)

[HTML Tags](#)

[Logic Tags](#)

[Nested Tags](#)

[Template Tags](#)

[Tiles Tags](#)

[Utilities](#)

[Validator](#)

## Quick Links

[Welcome](#)

[News and Status](#)

[Resources](#)

[User and  
Developer  
Guides \\*](#)

[FAQs and  
HowTos](#)

## 2. Building Model Components

Contributors:

- Craig R. McClanahan
- Mike Schachter
- Ted Husted
- Martin Cooper

### 2.1 Overview

Many requirements documents used for building web applications focus on the *View*. However, you should ensure that the processing required for each submitted request is also clearly defined from the *Model's* perspective. In general, the developer of the *Model* components will be focusing on the creation of JavaBeans classes that support all of the functional requirements. The precise nature of the beans required by a particular application will vary widely depending on those requirements, but they can generally be classified into several categories discussed below. However, a brief review of the concept of "scope" as it relates to beans and JSP is useful first.

### 2.2 JavaBeans and Scope

Within a web-based application, JavaBeans can be stored in (and accessed from) a number of different collections of "attributes". Each collection has different rules for the lifetime of that collection, and the visibility of the beans stored there. Together, the rules defining lifetime and visibility are called the *scope* of those beans. The JavaServer Pages (JSP) Specification defines scope choices using the following terms (with the equivalent servlet API concept defined in parentheses):

- **page** - Beans that are visible within a single JSP page, for the lifetime of the current request. (Local variables of the `service` method)
- **request** - Beans that are visible within a single JSP page, as well as to any page or servlet that is included in this page, or forwarded to by this page. (Request attributes)
- **session** - Beans that are visible to all JSP pages and servlets that participate in a particular user session, across one or more requests. (Session attributes)
- **application** - Beans that are visible to all JSP pages and servlets that are part of a web application. (Servlet context attributes)

It is important to remember that JSP pages and servlets in the same web application share the same sets of bean collections. For example, a bean stored as a request attribute in a servlet like this:

```
MyCart mycart = new MyCart(...);
request.setAttribute("cart", mycart);
```

is immediately visible to a JSP page which this servlet forwards to, using a standard action tag like this:

```
<jsp:useBean id="cart" scope="request"
class="com.mycompany.MyApp.MyCart" />
```

## 2.3 ActionForm Beans

**Note:** While ActionForm beans often have properties that correspond to properties in your Model beans, the form beans themselves should be considered a Controller component. As such, they are able to transfer data between the Model and View layers.

The Struts framework generally assumes that you have defined an ActionForm bean (that is, a Java class extending the ActionForm class) for the input forms in your application. ActionForm beans are sometimes just called "form beans". These may be finely-grained objects, so that there is one bean for each form, or coarsely-grained so that one bean serves several forms, or even an entire application.

If you declare such beans in your Struts configuration file (see "[Building the Controller Components](#)"), the Struts controller servlet will automatically perform the following services for you, before invoking the appropriate Action method:

- Check in the user's session for an instance of a bean of the appropriate class, under the appropriate key.
- If there is no such session scope bean available, a new one is automatically created and added to the user's session.
- For every request parameter whose name corresponds to the name of a property in the bean, the corresponding setter method will be called. This operates in a manner similar to the standard JSP action `<jsp:setProperty>` when you use the asterisk wildcard to select all properties.
- The updated ActionForm bean will be passed to the execute method of an Action class [org.apache.struts.Action], so that the values can be made available to your system state and business logic beans.

For more about coding Actions and ActionForm beans, see the "[Building Controller Components](#)" chapter.

You should note that a "form", in the sense discussed here, does not necessarily correspond to a single JSP page in the user interface. It is common in many applications to have a "form" (from the user's perspective) that extends over multiple pages. Think, for example, of the wizard style user interface that is commonly used when installing new applications. Struts encourages you to define a single ActionForm bean that contains properties for all of the fields, no matter which page the field is actually displayed on. Likewise, the various pages of the same form should all be submitted to the same Action Class. If you follow these suggestions, the page designers can rearrange the fields among the various pages, often without requiring changes to the processing logic.

Smaller applications may only need a single ActionForm to service all of its input forms. Others applications might use a single ActionForm for each major subsystem of the application. Some teams might prefer to have a separate ActionForm class for each distinct input form or workflow. How many or how few ActionForms to use is entirely up to you. The framework doesn't care.

## 2.4 System State Beans

The actual state of a system is normally represented as a set of one or more JavaBeans classes, whose properties define the current state. A shopping cart system, for example, will include a bean that represents the cart being maintained for each individual shopper, and will (among other things) include the set of items that the shopper has currently selected for purchase. Separately, the system will also include different beans for the user's profile information (including their credit card and ship-to addresses), as well as the catalog of available items and their current inventory levels.

For small scale systems, or for state information that need not be kept for a long period of time, a set of system state beans may contain all the knowledge that the system ever has of these particular details. Or, as is often the case, the system state beans will represent information that is stored permanently in some external database (such as a `CustomerBean` object that corresponds to a particular row in the CUSTOMERS table), and are created or removed from the server's memory as needed. Entity Enterprise JavaBeans are also used for this purpose in large scale applications.

## 2.5 Business Logic Beans

You should encapsulate the functional logic of your application as method calls on JavaBeans designed for this purpose. These methods may be part of the same classes used for the system state beans, or they may be in separate classes dedicated to performing the logic. In the latter case, you will usually need to pass the system state beans to be manipulated to these methods as arguments.

For maximum code re-use, business logic beans should be designed and implemented so that they do **not** know they are being executed in a web application environment. If you find yourself having to import a `javax.servlet.*` class in your bean, you are tying this business logic to the web application environment. Consider rearranging things so that your `Action` classes (part of the Controller role, as described below) translate all required information from the HTTP request being processed into property setter calls on your business logic beans, after which a call to an `execute` method can be made. Such a business logic class can be reused in environments other than the web application for which they were initially constructed.

Depending on the complexity and scope of your application, business logic beans might be ordinary JavaBeans that interact with system state beans passed as arguments, or ordinary JavaBeans that access a database using JDBC calls. For larger applications, these beans will often be stateful or stateless Enterprise JavaBeans (EJBs) instead.

For more about using a database with your application, see the [Accessing a Database HowTo](#).

Next: [Building View Components](#)





## User Guide

[Table of](#)
[Contents](#)
[Preface](#)
[Introduction](#)
[Model](#)
[Components](#)
[View](#)
[Components](#)
[Controller](#)
[Components](#)
[Configuration](#)
[Release](#)
[Notes](#)
[Installation](#)

## Developer Guides

[Bean Tags](#)
[HTML](#)
[Tags](#)
[Logic](#)
[Tags](#)
[Nested](#)
[Tags](#)
[Template](#)
[Tags](#)
[Tiles Tags](#)
[Utilities](#)
[Validator](#)

## Quick Links

[Welcome](#)
[News and](#)
[Status](#)
[Resources](#)
[User and](#)
[Developer](#)
[Guides \\*](#)
[FAQs and](#)
[HowTos](#)

## 3. Building View Components

Contributors:

- Craig R. McClanahan
- Mike Schachter
- Ted Husted
- Martin Cooper
- Ed Burns
- James DeVries
- David Graham

### 3.1 Overview

This chapter focuses on the task of building the *View* components for use with the Struts framework. Many applications rely on JavaServer Pages (JSP) technology to create the presentation layer. The Struts distribution includes a comprehensive JSP tag library that provides support for building internationalized applications, as well as for interacting with input forms. Several other topics related to the View components are briefly discussed.

### 3.2 Internationalized Messages

A few years ago, application developers could count on having to support only residents of their own country, who are used to only one (or sometimes two) languages, and one way to represent numeric quantities like dates, numbers, and monetary values. However, the explosion of application development based on web technologies, as well as the deployment of such applications on the Internet and other broadly accessible networks, have rendered national boundaries invisible in many cases. This has translated (if you will pardon the pun) into a need for applications to support *internationalization* (often called "i18n" because 18 is the number of letters in between the "i" and the "n") and *localization*.

Struts builds upon the standard classes available on the Java platform to build internationalized and localized applications. The key concepts to become familiar with are:

- **[Locale](#)** - The fundamental Java class that supports internationalization is `Locale`. Each `Locale` represents a particular choice of country and language (plus an optional language variant), and also a set of formatting assumptions for things like numbers and dates.
- **[ResourceBundle](#)** - The `java.util.ResourceBundle` class provides the fundamental tools for supporting messages in multiple languages. See the Javadocs for the `ResourceBundle` class, and the information on Internationalization in the documentation bundle for your JDK release, for more information.
- **[PropertyResourceBundle](#)** - One of the standard implementations of `ResourceBundle` allows you to define resources using the same "name=value" syntax used to initialize properties files. This is very convenient for preparing resource bundles with messages that are used in a web application, because these messages are generally text oriented.
- **[MessageFormat](#)** - The `java.text.MessageFormat` class allows you to replace portions of a message string (in this case, one retrieved from a resource bundle) with arguments specified at run time. This is useful in cases where you are creating a sentence, but the words would appear in a



different order in different languages. The placeholder string `{0}` in the message is replaced by the first runtime argument, `{1}` is replaced by the second argument, and so on.

- **[MessageResources](#)** - The Struts class `org.apache.struts.util.MessageResources` lets you treat a set of resource bundles like a database, and allows you to request a particular message string for a particular Locale (normally one associated with the current user) instead of for the default Locale the server itself is running in.

Please note that the i18n support in a framework like Struts is limited to the **presentation** of internationalized text and images to the user. Support for Locale specific **input methods** (used with languages such as Japanese, Chinese, and Korean) is left up to the client device, which is usually a web browser.

For an internationalized application, follow the steps described in the Internationalization document in the JDK documentation bundle for your platform to create a properties file containing the messages for each language. An example will illustrate this further:

Assume that your source code is created in package `com.mycompany.mypackage`, so it is stored in a directory (relative to your source directory) named `com/mycompany/mypackage`. To create a resource bundle called `com.mycompany.mypackage.MyApplication`, you would create the following files in the `com/mycompany/mypackage` directory:

- **MyApplication.properties** - Contains the messages in the default language for your server. If your default language is English, you might have an entry like this: `prompt.hello=Hello`
- **MyApplication\_xx.properties** - Contains the same messages in the language whose ISO language code is "xx" (See the [ResourceBundle Javadoc](#) page for a link to the current list). For a French version of the message shown above, you would have this entry: `prompt.hello=Bonjour` You can have resource bundle files for as many languages as you need.

When you configure the controller servlet in the web application deployment descriptor, one of the things you will need to define in an initialization parameter is the base name of the resource bundle for the application. In the case described above, it would be `com.mycompany.mypackage.MyApplication`.

```
<servlet>
<servlet-name>action</servlet-name>
<servlet-class>
org.apache.struts.action.ActionServlet
</servlet-class>
<init-param>
<param-name>application</param-name>
<param-value>
    com.mycompany.mypackage.MyResources
</param-value>
</init-param>
<!-- ... -->
</servlet>
```

The important thing is for the resource bundle to be found on the class path for your application. Another approach is to store the `MyResources.properties` file in your application's `classes` folder. You can then simply specify "myResources" as the application value. Just be careful it is not deleted if your build script deletes classes as part of a "clean" target.

If it does, here is an Ant task to run when compiling your application that copies the contents of a `src/conf` directory to the `classes` directory:

```
<!-- Copy any configuration files -->
<copy todir="classes">
<fileset dir="src/conf"/>
```

&lt;/copy&gt;

### 3.3 Forms and FormBean Interactions

**Note:** While the examples given here use JSP and custom tags, the ActionForm beans and the other Struts controller components are View neutral. Struts can be used with Velocity Templates, XSL, and any other presentation technology that can be rendered via a Java servlet. See the [Resources page](#) for links to additional information.

At one time or another, most web developers have built forms using the standard capabilities of HTML, such as the `<input>` tag. Users have come to expect interactive applications to have certain behaviors, and one of these expectations relates to error handling -- if the user makes an error, the application should allow them to fix just what needs to be changed -- without having to re-enter any of the rest of the information on the current page or form.

Fulfilling this expectation is tedious and cumbersome when coding with standard HTML and JSP pages. For example, an input element for a username field might look like this (in JSP):

```
<input type="text" name="username"
value="<%= loginBean.getUsername() %>" />
```

which is difficult to type correctly, confuses HTML developers who are not knowledgeable about programming concepts, and can cause problems with HTML editors. Instead, Struts provides a comprehensive facility for building forms, based on the Custom Tag Library facility of JSP 1.1. The case above would be rendered like this using Struts:

```
<html:text property="username" />;
```

with no need to explicitly refer to the JavaBean from which the initial value is retrieved. That is handled automatically by the JSP tag, using facilities provided by the framework.

HTML forms are sometimes used to upload other files. Most browsers support this through a `<input type="file">` element, that generates a file browse button, but it's up to the developer to handle the incoming files. Struts handles these "multipart" forms in a way identical to building normal forms.

For an example of using Struts to create a simple login form, see the "[Building an ActionForm Howto](#)".

#### 3.3.1 Indexed & Mapped Properties

Property references in JSP pages using the Struts framework can reference Java Bean properties as described in the JavaBeans specification. Most of these references refer to "scalar" bean properties, referring to primitive or single Object properties. However, Struts, along with the Jakarta Commons Beanutils library, allow you to use property references which refer to individual items in an array, collection, or map, which are represented by bean methods using well-defined naming and signature schemes.

Documentation on the Beanutils package can be found at <http://jakarta.apache.org/commons/beanutils/api/index.html>. More information about using indexed and mapped properties in Struts can be found in the FAQ describing [Indexed Properties](#), [Mapped Properties](#), and [Indexed Tags](#).

#### 3.3.2 Input Field Types Supported

Struts defines HTML tags for all of the following types of input fields, with hyperlinks to the corresponding reference information.

- [checkboxes](#)
- [hidden](#) fields
- [password](#) input fields
- [radio](#) buttons
- [reset](#) buttons
- [select](#) lists with embedded option or options items
- [option](#)
- [options](#)
- [submit](#) buttons
- [text](#) input fields
- [textareas](#)

In every case, a field tag must be nested within a `form` tag, so that the field knows what bean to use for initializing displayed values.

### 3.3.3 Other Useful Presentation Tags

There are several tags useful for creating presentations, consult the documentation on each specific tag library, along with the Tag Developers Guides, for more information:

- [logic] [iterate](#) repeats its tag body once for each element of a specified collection (which can be an Enumeration, a Hashtable, a Vector, or an array of objects).
- [logic] [present](#) depending on which attribute is specified, this tag checks the current request, and evaluates the nested body content of this tag only if the specified value is present. Only one of the attributes may be used in one occurrence of this tag, unless you use the property attribute, in which case the name attribute is also required. The attributes include cookie, header, name, parameter, property, role, scope, and user.
- [logic] [notPresent](#) the companion tag to present, notPresent provides the same functionality when the specified attribute is not present.
- [html] [link](#) generates a HTML `<a>` element as an anchor definition or a hyperlink to the specified URL, and automatically applies URL encoding to maintain session state in the absence of cookie support.
- [html] [img](#) generates a HTML `<img>` element with the ability to dynamically modify the URLs specified by the "src" and "lowsrc" attributes in the same manner that `<html:link>` can.
- [bean] [parameter](#) retrieves the value of the specified request parameter, and defines the result as a page scope attribute of type String or String[].

### 3.3.4 Automatic Form Validation

In addition to the form and bean interactions described above, Struts offers an additional facility to validate the input fields it has received. To utilize this feature, override the following method in your `ActionForm` class:

```
validate(ActionMapping mapping,
HttpServletRequest request);
```

The `validate` method is called by the controller servlet after the bean properties have been populated, but before the corresponding action class's `perform` method is invoked. The `validate` method has the following options:

- Perform the appropriate validations and find no problems -- Return either `null` or a zero-length `ActionErrors` instance, and the controller servlet will proceed to call the `perform` method of the appropriate `Action` class.
- Perform the appropriate validations and find problems -- Return an `ActionErrors` instance containing `ActionError`'s, which are classes that contain the error message keys (into the application's `MessageResources` bundle) that should be displayed. The controller servlet will store this array as a request attribute suitable for use by the `<html:errors>` tag, and will forward control back to the input form (identified by the `input` property for this `ActionMapping`).

As mentioned earlier, this feature is entirely optional. The default implementation of the `validate` method returns `null`, and the controller servlet will assume that any required validation is done by the action class.

One common approach is to perform simple, *prima facie* validations using the `ActionForm` `validate` method, and then handle the "business logic" validation from the `Action`.

The Struts Validator, covered in the next section, may be used to easily validate `ActionForms`.

### 3.3.5 The Struts Validator

Configuring the Validator to perform form validation is easy.

1. The `ActionForm` bean must extend `ValidatorForm`
2. The form's JSP must include the `<html:javascript>` tag for client side validation.
3. You must define the validation rules in an xml file like this:

```
<form-validation>
<formset>
<form name="logonForm">
  <field property="username" depends="required">
    <msg name="required" key="error.username"/>
  </field>
</form>
</formset>
</form-validation>
```

The `msg` element points to the message resource key to use when generating the error message.

4. Lastly, you must enable the `ValidatorPlugin` in the `struts-config.xml` file like this:

```
<plug-in className="org.apache.struts.validator.ValidatorPlugIn">
  <set-property
property="pathnames"
value="/WEB-INF/validator-rules.xml,
  /WEB-INF/validation.xml"/>
```

```
</plug-in>
```

For more about the Struts Validator, see the [Developers Guide](#).

## 3.4 Other Presentation Techniques

Although the look and feel of your application can be completely constructed based on the standard capabilities of JSP and the Struts custom tag library, you should consider employing other techniques that will improve component reuse, reduce maintenance efforts, and/or reduce errors. Several options are discussed in the following sections.

### 3.4.1 Application-Specific Custom Tags

Beyond using the custom tags provided by the Struts library, it is easy to create tags that are specific to the application you are building, to assist in creating the user interface. The MailRead example application included with Struts illustrates this principle by creating the following tags unique to the implementation of this application:

- **checkLogon** - Checks for the existence of a particular session object, and forwards control to the logon page if it is missing. This is used to catch cases where a user has bookmarked a page in the middle of your application and tries to bypass logging on, or if the user's session has been timed out.
- **linkSubscription** - Generates a hyperlink to a details page for a Subscription, which passes the required primary key values as request attributes. This is used when listing the subscriptions associated with a user, and providing links to edit or delete them.
- **linkUser** - Generates a hyperlink to a details page for a User, which passes the required primary key values as request attributes.

The source code for these tags is in the `src/example` directory, in package `org.apache.struts.example`, along with the other Java classes that are used in this application.

### 3.4.2 Page Composition With Includes

Creating the entire presentation of a page in one JSP file (with custom tags and beans to access the required dynamic data) is a very common design approach, and was employed in the example application included with Struts. However, many applications require the display of multiple logically distinct portions of your application together on a single page.

For example, a portal application might have some or all of the following functional capabilities available on the portal's "home" page:

- Access to a search engine for this portal.
- One or more "news feed" displays, with the topics of interest customized from the user's registration profile.
- Access to discussion topics related to this portal.
- A "mail waiting" indicator if your portal provides free email accounts.

The development of the various segments of this site is easier if you can divide up the work, and assign different developers to the different segments. Then, you can use the *include* capability of JavaServer Pages technology to combine the results into a single result page, or use the include tag provided with Struts. There are three types of *include* available, depending on when you want the combination of output to occur:

- An `<%@ include file="xxxxx" %>` directive can include a file that contains Java code or JSP tags. The code in the included file can even reference variables declared earlier in the outer jsp

page. The code is inlined into the other JavaServer Page before it is compiled so it can definitely contain more than just HTML.

- The include *action* (`<jsp:include page="xxxxx" flush="true" />`) is processed at request time, and is handled transparently by the server. Among other things, that means you can conditionally perform the include by nesting it within a tag like [equals](#) by using its parameter attribute.
- The [bean:include](#) tag takes either an argument "forward" representing a logical name mapped to the jsp to include, or the "id" argument, which represents a page context String variable to print out to the jsp page.

### 3.4.3 Page Composition With Tiles

Tiles is a powerful templating library that allows you to construct views by combining various "tiles". Here's a quick setup guide:

1. Create a `/layout/layout.jsp` file that contains your app's common look and feel:

```
<html>
<body>
<tiles:insert attribute="body" />
</body>
</html>
```

2. Create your `/index.jsp` homepage file:

```
<h1>This is my homepage!</h1>
```

3. Create a `/WEB-INF/tiles-defs.xml` file that looks like this:

```
<tiles-definitions>
<definition
  name="layout"
  path="/layout/layout.jsp">
  <put name="body" value="" />
</definition>
<definition name="homepage" extends="layout">
  <put
    name="body"
    value="/index.jsp" />
</definition>
</tiles-definitions>
```

4. Setup the TilesPlugin in the `struts-config.xml` file:

```
<plug-in
  className="org.apache.struts.tiles.TilesPlugin">
  <set-property
    property="definitions-config"
    value="/WEB-INF/tiles-defs.xml" />
</plug-in>
```

5. Setup an action mapping in `struts-config.xml` to point to your homepage tile:

```
<action
  path="/index"
  type="org.apache.struts.actions.ForwardAction"
```

```
parameter="homepage" />
```

The TilesPlugin configures a special RequestProcessor that determines if the requested view is a tile and processes it accordingly. Note that we made the homepage tile extend our root layout tile and changed the body attribute. Tiles inserts the file named in the body attribute into the main layout.

See the tiles-documentation webapp for in-depth examples.

### 3.4.4 Image Rendering Components

Some applications require dynamically generated images, like the price charts on a stock reporting site. Two different approaches are commonly used to meet these requirements:

- Render a hyperlink with a URL that executes a servlet request. The servlet will use a graphics library to render the graphical image, set the content type appropriately (such as `image/gif`), and send back the bytes of that image to the browser, which will display them just as if it had received a static file.
- Render the HTML code necessary to download a Java applet that creates the required graph. You can configure the graph by setting appropriate initialization parameters for the applet in the rendered code, or you can have the applet make its own connection to the server to receive these parameters.

### 3.4.5 Rendering Text

Some applications require dynamically generated text or markup, such as XML. If a complete page is being rendered, and can be output using a `PrintWriter`, this is very easy to do from an Action:

```
response.setContentType("text/plain"); // or text/xml
PrintWriter writer = response.getWriter();
// use writer to render text
return(null);
```

### 3.4.6 The Struts-EL Tag Library

The **Struts-EL** tag library is a contributed library in the Struts distribution. It represents an integration of the Struts tag library with the JavaServer Pages Standard Tag Library, or at least the "expression evaluation" engine that is used by the JSTL.

The base Struts tag library contains tags which rely on the evaluation of "rtexprvalue"s (runtime scriptlet expressions) to evaluate dynamic attribute values. For instance, to print a message from a properties file based on a resource key, you would use the `bean:write` tag, perhaps like this:

```
<bean:message key='<%= stringvar %>' />
```

This assumes that `stringvar` exists as a JSP scripting variable. If you're using the **Struts-EL** library, the reference looks very similar, but slightly different, like this:

```
<bean-el:message key="{stringvar}" />
```

If you want to know how to properly use the **Struts-EL** tag library, there are two important things you need to know:

- The Struts tag library

- The JavaServer Pages Standard tag library

Once you understand how to use these two, consider Struts tag attribute values being evaluated the same way the JSTL tag attribute values are. Past that, there is very little else you need to know to effectively use the **Struts-EL** tag library.

Although the **Struts-EL** tag library is a direct "port" of the tags from the Struts tag library, not all of the tags in the Struts tag library were implemented in the **Struts-EL** tag library. This was the case if it was clear that the functionality of a particular Struts tag could be entirely fulfilled by a tag in the JSTL. It is assumed that developers will want to use the **Struts-EL** tag library along with the JSTL, so it is reasonable to assume that they will use tags from the JSTL if they fill their needs.

Next: [Building Controller Components](#)

---

*Copyright (c) 2000-2002, Apache Software Foundation*

Powered by  
**Struts**





## User Guide

[Table of Contents](#)  
[Preface](#)  
[Introduction](#)  
[Model Components](#)  
[View Components](#)  
[Controller Components](#)  
[Configuration](#)  
[Release Notes](#)  
[Installation](#)

## Developer Guides

[Bean Tags](#)  
[HTML Tags](#)  
[Logic Tags](#)  
[Nested Tags](#)  
[Template Tags](#)  
[Tiles Tags](#)  
[Utilities](#)  
[Validator](#)

## Quick Links

[Welcome](#)  
[News and Status](#)  
[Resources](#)  
[User and Developer Guides \\*](#)  
[FAQs and HowTos](#)

## 4. Building Controller Components

Contributors:

- Craig R. McClanahan
- Mike Schachter
- Ted Husted
- Martin Cooper
- Ed Burns
- Donald Ball
- Eddie Bush
- Yann Cebron
- David Graham
- Tim O'Brien

### 4.1 Overview

Now that we understand how to construct the Model and View components of your application, it is time to focus on the Controller components. Struts includes a servlet that implements the primary function of mapping a request URI to an Action class. Therefore, your primary responsibilities related to the Controller are:

- Write an `ActionForm` class to mediate between the Model and the View, as described in [Building Model Components](#).
- Write an `Action` class for each logical request that may be received (extend `org.apache.struts.action.Action`).
- Configure a `ActionMapping` (in XML) for each logical request that may be submitted. The XML configuration file is usually named `struts-config.xml`.

To deploy your application, you will also need to:

- Update the web application deployment descriptor file (in XML) for your application to include the necessary Struts components.
- Add the appropriate Struts components to your application.

The latter two items are covered in the "[Configuring Applications](#)" chapter.

### 4.2 The ActionServlet

For those of you familiar with MVC architecture, the `ActionServlet` represents the C - the controller. The job of the controller is to:

- process user requests,
- determine what the user is trying to achieve according to the request,
- pull data from the model (if necessary) to be given to the appropriate view, and
- select the proper view to respond to the user.

The Struts controller delegates most of this grunt work to Action classes.

In addition to being the controller for your application, the `ActionServlet` instance also is responsible for initialization and clean-up of resources. When the controller initializes, it first loads the application config corresponding to the "config" init-param. It then goes through an enumeration of all `init-param` elements, looking for those elements whose name starts with `config/`. For each of these elements, Struts loads the configuration file specified by the value of that `init-param`, and assigns a "prefix" value to that module's `ApplicationConfig` instance consisting of the piece of the `init-param` name following "config/". For example, the module prefix specified by the `init-param config/foo` would be "foo". This is important to know, since this is how the controller determines which module will be given control of processing the request. To access the module foo, you would use a URL like:

```
http://localhost:8080/myApp/foo/someAction.do
```

For each request made of the controller, the method `process(HttpServletRequest, HttpServletResponse)` will be called. This method simply determines which module should service the request and then invokes that module's `RequestProcessor`'s `process` method, passing the same request and response.

## 4.2.1 Request Processor

The `RequestProcessor` is where the majority of the core processing occurs for each request. Let's take a look at the helper functions the `process` method invokes in-turn:

|                                |   |
|--------------------------------|---|
| <code>processPath</code>       | Determine the path that invoked us. This will be used later to retrieve an <code>ActionMapping</code> .   |
| <code>processLocale</code>     | Select a locale for this request, if one hasn't already been selected, and place it in the request.   |
| <code>processContent</code>    | Set the default content type (with optional character encoding) for all responses if requested.   |
| <code>processNoCache</code>    | If appropriate, set the following response headers: "Pragma", "Cache-Control", and "Expires".   |
| <code>processPreprocess</code> | This is one of the "hooks" the <code>RequestProcessor</code> makes available for subclasses to override. The default implementation simply returns <code>true</code> . If you subclass <code>RequestProcessor</code> and override <code>processPreprocess</code> you should either return <code>true</code> (indicating process should continue processing the request) or <code>false</code> (indicating you have handled the request and the process should return) |
| <code>processMapping</code>    | Determine the <code>ActionMapping</code> associated with this path.   |

|                                   |   |
|-----------------------------------|---|
| <code>processRoles</code>         | If the mapping has a role associated with it, ensure the requesting user is has the specified role. If they do not, raise an error and stop processing of the request.  |
| <code>processActionForm</code>    | Instantiate (if necessary) the <code>ActionForm</code> associated with this mapping (if any) and place it into the appropriate scope.   |
| <code>processPopulate</code>      | Populate the <code>ActionForm</code> associated with this request, if any.  |
| <code>processValidate</code>      | Perform validation (if requested) on the <code>ActionForm</code> associated with this request (if any).   |
| <code>processForward</code>       | If this mapping represents a forward, forward to the path specified by the mapping.   |
| <code>processInclude</code>       | If this mapping represents an include, include the result of invoking the path in this request.   |
| <code>processActionCreate</code>  | Instantiate an instance of the class specified by the current <code>ActionMapping</code> (if necessary).  |
| <code>processActionPerform</code> | This is the point at which your action's <code>perform</code> or <code>execute</code> method will be called.  |
| <code>processActionForward</code> | Finally, the <code>process</code> method of the <code>RequestProcessor</code> takes the <code>ActionForward</code> returned by your <code>Action</code> class, and uses to select the next resource (if any). Most often the <code>ActionForward</code> leads to the presentation page that renders the response. |

## 4.3 ActionForm Classes

An `ActionForm` represents an HTML form that the user interacts with over one or more pages. You will provide properties to hold the state of the form with getters and setters to access them. `ActionForms` can be stored in either the session (default) or request scopes. If they're in the session it's important to implement the form's `reset` method to initialize the form before each use. Struts sets the `ActionForm`'s properties from the request parameters and sends the validated form to the appropriate `Action`'s `execute` method.

When you code your `ActionForm` beans, keep the following principles in mind:

- The `ActionForm` class itself requires no specific methods to be implemented. It is used to identify the role these particular beans play in the overall architecture. Typically, an `ActionForm` bean will have only property getter and property setter methods, with no business logic.
- The `ActionForm` object also offers a standard validation mechanism. If you override a "stub" method, and provide error messages in the standard application resource, Struts will automatically validate the input from the form (using your method). See "[Action Form Validation](#)" for details. Of course, you can also ignore the `ActionForm` validation and provide your own in the `Action` object.
- Define a property (with associated `getXxx` and `setXxx` methods) for each field that is present in the form. The field name and property name must match according to the usual JavaBeans conventions (see the Javadoc for the `java.beans.Introspector` class for a start on information about this). For example, an input field named `username` will cause the `setUsername` method to be called.
- Buttons and other controls on your form can also be defined as properties. This can help determine which button or control was selected when the form was submitted. Remember, the `ActionForm` is meant to represent your data-entry form, not just the data beans.
- Think of your `ActionForm` beans as a firewall between HTTP and the `Action`. Use the `validate` method to ensure all required properties are present, and that they contain

reasonable values. An ActionForm that fails validation will not even be presented to the Action for handling.

- You may also place a bean instance on your form, and use nested property references. For example, you might have a "customer" bean on your ActionForm, and then refer to the property "customer.name" in your presentation page. This would correspond to the methods `customer.getName()` and `customer.setName(string Name)` on your customer bean. See the Tag Library Developer Guides for more about using nested syntax with the Struts JSP tags.
- *Caution:* If you nest an existing bean instance on your form, think about the properties it exposes. Any public property on an ActionForm that accepts a single String value can be set with a query string. It may be useful to place beans that can affect the business state inside a thin "wrapper" that exposes only the properties required. This wrapper can also provide a filter to be sure runtime properties are not set to inappropriate values.

### 4.3.1 DynaActionForm Classes

Maintaining a separate concrete ActionForm class for each form in your Struts application is time-consuming. This can be alleviated through the use of DynaActionForm classes. Instead of creating a new ActionForm subclass and new get/set methods for each of your bean's properties, you can list its properties, type, and defaults in the Struts configuration file.

For example, add the following to `struts-config.xml` for a UserForm bean that stores a user's given and family names:

```
<form-bean
  name="UserForm"
  type="org.apache.struts.action.DynaActionForm">
  <form-property
    name="givenName"
    type="java.lang.String"
    initial="John" />
  <form-property
    name="familyName"
    type="java.lang.String"
    initial="Smith" />
</form-bean>
```

The types supported by DynaActionForm include:

- `java.lang.BigDecimal`
- `java.lang.BigInteger`
- `boolean` and `java.lang.Boolean`
- `byte` and `java.lang.Byte`
- `char` and `java.lang.Character`
- `java.lang.Class`
- `double` and `java.lang.Double`
- `float` and `java.lang.Float`
- `int` and `java.lang.Integer`
- `long` and `java.lang.Long`
- `short` and `java.lang.Short`
- `java.lang.String`
- `java.sql.Date`
- `java.sql.Time`
- `java.sql.Timestamp`

If you do not supply an initial attribute, numbers will be initialized to 0 and objects to null.

In JSP pages using Struts custom tags, attributes of `DynaActionForm` objects can be referenced just like ordinary `ActionForm` objects. However, when utilizing the JavaServer Pages Standard Tag Library, only properties of ordinary `ActionForm` objects can be directly accessed through the JSTL expression language syntax. However, `DynaActionForm` properties can be accessed through a slightly different syntax. In particular, whereas the JSTL EL syntax for referencing a property of an `ActionForm` would be something like:

```
${formbean.prop}
```

The syntax for referencing a property of a `DynaActionForm` would be:

```
${dynabean.map.prop}
```

The map property is a property of `DynaActionForm` which represents the `HashMap` containing the `DynaActionForm` properties.

### 4.3.2 Map-backed ActionForms

The `DynaActionForm` classes offer the ability to create `ActionForm` beans at initialization time, based on a list of properties enumerated in the Struts configuration file. However, many HTML forms are generated dynamically at request time. Since the properties of these forms' `ActionForm` beans are not all known ahead of time, we need a new approach.

Struts allows you to make one or more of your `ActionForm`'s properties' values a `Map` instead of a traditional atomic object. You can then store the data from your form's dynamic fields in that `Map`. Here is an example of a map-backed `ActionForm` class:

```
public FooForm extends ActionForm {

    private final Map values = new HashMap();

    public void setValue(String key, Object value) {
        values.put(key, value);
    }

    public Object getValue(String key) {
        return values.get(key);
    }

}
```

In its corresponding JSP page, you can access objects stored in the values map using a special notation: `mapname (keyname)`. The parentheses in the bean property name indicate that:

- the bean property named `mapname` is indexed using Strings (probably backed by a `Map`), and that
- Struts should look for get/set methods that take a `String` key parameter to find the correct sub-property value. Struts will, of course, use the `keyname` value from the parentheses when it calls the get/set methods.

Here is a simple example:

```
<html:text property="value(foo)"/>
```

This will call the `getValue` method on `FooForm` with a key value of `"foo"` to find the property value. To create a form with dynamic field names, you could do the following:

```
<% for (int i=0; i<10; i++) {  
String name = "value(foo-" + i + ")";  
<html:text property="<%=name%>" /><br/>  
%>
```

Note that there is nothing special about the name value. Your map-backed property could instead be named `property`, `thingy`, or any other bean property name you prefer. You can even have multiple map-backed properties on the same bean.

In addition to map-backed properties, you can also create list-backed properties. You do so by creating indexed `get/set` methods on your bean:

```
public FooForm extends ActionForm {  
  
    private final List values = new ArrayList();  
  
    public void setValue(int key, Object value) {  
        values.set(key, value);  
    }  
  
    public Object getValue(int key) {  
        return values.get(key);  
    }  
}
```

In your presentation pages, you access individual entries in a list-backed property by using a different special notation: `listname[index]`. The braces in the bean property name indicate that the bean property named `listname` is indexed (probably backed by a `List`), and that Struts should look for `get/set` methods that take an index parameter in order to find the correct sub-property value.

## 4.4 Action Classes

The `Action` class defines two methods that could be executed depending on your servlet environment:

```
public ActionForward execute(ActionMapping mapping,  
                             ActionForm form,  
                             ServletRequest request,  
                             ServletResponse response)  
throws Exception;  
  
public ActionForward execute(ActionMapping mapping,  
                             ActionForm form,  
                             HttpServletRequest request,  
                             HttpServletResponse response)  
throws Exception;
```

Since the majority of Struts projects are focused on building web applications, most projects will only use the `"HttpServletRequest"` version. A non-HTTP `execute()` method has been provided for

applications that are not specifically geared towards the HTTP protocol.

The goal of an `Action` class is to process a request, via its `execute` method, and return an `ActionForward` object that identifies where control should be forwarded (e.g. a JSP, Tile definition, Velocity template, or another `Action`) to provide the appropriate response. In the *MVC/Model 2* design pattern, a typical `Action` class will often implement logic like the following in its `execute` method:

- Validate the current state of the user's session (for example, checking that the user has successfully logged on). If the `Action` class finds that no logon exists, the request can be forwarded to the presentation page that displays the username and password prompts for logging on. This could occur because a user tried to enter an application "in the middle" (say, from a bookmark), or because the session has timed out, and the servlet container created a new one.
- If validation is not complete, validate the form bean properties as needed. If a problem is found, store the appropriate error message keys as a request attribute, and forward control back to the input form so that the errors can be corrected.
- Perform the processing required to deal with this request (such as saving a row into a database). This *can* be done by logic code embedded within the `Action` class itself, **but** should generally be performed by calling an appropriate method of a business logic bean.
- Update the server-side objects that will be used to create the next page of the user interface (typically request scope or session scope beans, depending on how long you need to keep these items available).
- Return an appropriate `ActionForward` object that identifies the presentation page to be used to generate this response, based on the newly updated beans. Typically, you will acquire a reference to such an object by calling `findForward` on either the `ActionMapping` object you received (if you are using a logical name local to this mapping), or on the controller servlet itself (if you are using a logical name global to the application).

In Struts 1.0, `Actions` called a `perform` method instead of the now-preferred `execute` method. These methods use the same parameters and differ only in which exceptions they throw. The elder `perform` method throws `ServletException` and `IOException`. The new `execute` method simply throws `Exception`. The change was to facilitate the Declarative Exception handling feature introduced in Struts 1.1.

The `perform` method may still be used in Struts 1.1 but is deprecated. The Struts 1.1 method simply calls the new `execute` method and wraps any `Exception` thrown as a `ServletException`.

## 4.4.1 Action Class Design Guidelines

Remember the following design guidelines when coding `Action` classes:

- **Write code for a multi-threaded environment** - The controller servlet creates **only one instance of your `Action` class**, and uses this one instance to service all requests. Thus, you need to write thread-safe `Action` classes. Follow the same guidelines you would use to write thread-safe Servlets. Here are two general guidelines that will help you write scalable, thread-safe `Action` classes:
  - **Only Use Local Variables** - The most important principle that aids in thread-safe coding is to use only local variables, **not instance variables**, in your `Action` class. Local variables are created on a stack that is assigned (by your JVM) to each request thread, so there is no need to worry about sharing them. An `Action` can be factored into several local methods, so long as all variables needed are passed as method parameters. This assures thread safety, as the JVM handles such variables



internally using the call stack which is associated with a single Thread.

- **Conserve Resources** - As a general rule, allocating scarce resources and keeping them across requests from the same user (in the user's session) can cause scalability problems. For example, if your application uses JDBC and you allocate a separate JDBC connection for every user, you are probably going to run in some scalability issues when your site suddenly shows up on Slashdot. You should strive to use pools and release resources (such as database connections) prior to forwarding control to the appropriate View component -- even if a bean method you have called throws an exception.
- **Don't throw it, catch it!** - Ever used a commercial website only to have a stack trace or exception thrown in your face after you've already typed in your credit card number and clicked the purchase button? Let's just say it doesn't inspire confidence. Now is your chance to deal with these application errors - in the `Action` class. If your application specific code throws exceptions you should catch these exceptions in your `Action` class, log them in your application's log (`servlet.log("Error message", exception)`) and return the appropriate `ActionForward`.

It is wise to avoid creating lengthy and complex `Action` classes. If you start to embed too much logic in the `Action` class itself, you will begin to find the `Action` class hard to understand, maintain, and impossible to reuse. Rather than creating overly complex `Action` classes, it is generally a good practice to move most of the persistence, and "business logic" to a separate application layer. When an `Action` class becomes lengthy and procedural, it may be a good time to refactor your application architecture and move some of this logic to another conceptual layer; otherwise, you may be left with an inflexible application which can only be accessed in a web-application environment. Struts should be viewed as simply the **foundation** for implementing MVC in your applications. Struts provides you with a useful control layer, but it is not a fully featured platform for building MVC applications, soup to nuts.

The example application included with Struts stretches this design principle somewhat, because the business logic itself is embedded in the `Action` classes. This should be considered something of a bug in the design of the example, rather than an intrinsic feature of the Struts architecture, or an approach to be emulated.

## 4.5 Exception Handler

You can define an `ExceptionHandler` to execute when an `Action`'s `execute` method throws an `Exception`. First, you need to subclass `org.apache.struts.action.ExceptionHandler` and override the `execute` method. Your `execute` method should process the `Exception` and return an `ActionForward` object to tell Struts where to forward to next. Then you configure your handler in `struts-config.xml` like this:

```
<global-exceptions>
  <exception
    key="some.key"
    type="java.io.IOException"
    handler="com.yourcorp.ExceptionHandler">
</global-exceptions>
```

This configuration element says that `com.yourcorp.ExceptionHandler.execute` will be called when any `IOException` is thrown by an `Action`. The key is a key into your message resources properties file that can be used to retrieve an error message.

You can override global exception handlers by defining a handler inside an action definition.



A common use of `ExceptionHandler`s is to configure one for `java.lang.Exception` so it's called for any exception and log the exception to some data store.

## 4.6 PlugIn Classes

The *PlugIn* interface extends *Action* and so that applications can easily hook into the *ActionServlet* lifecycle. This interface defines two methods, `init()` and `destroy()`, which are called at application startup and shutdown, respectively. A common use of a *PlugIn* *Action* is to configure or load application-specific data as the web application is starting up.

At runtime, any resource setup by `init` would be accessed by *Actions* or business tier classes. The *PlugIn* interface allows you to setup resources, but does not provide any special way to access them. Most often, the resource would be stored in application context, under a known key, where other components can find it.

*PlugIns* are configured using `<plug-in>` elements within the Struts configuration file. See [PlugIn Configuration](#) for details.

## 4.7 The ActionMapping Implementation

In order to operate successfully, the Struts controller servlet needs to know several things about how each request URI should be mapped to an appropriate *Action* class. The required knowledge has been encapsulated in a Java class named *ActionMapping*, the most important properties are as follows:

- `type` - Fully qualified Java class name of the *Action* implementation class used by this mapping.
- `name` - The name of the form bean defined in the config file that this action will use.
- `path` - The request URI path that is matched to select this mapping. See below for examples of how matching works.
- `unknown` - Set to `true` if this action should be configured as the default for this application, to handle all requests not handled by another action. Only one action can be defined as a default within a single application.
- `validate` - Set to `true` if the `validate` method of the action associated with this mapping should be called.
- `forward` - The request URI path to which control is passed when this mapping is invoked. This is an alternative to declaring a `type` property.

## 4.8 Writing Action Mappings

How does the controller servlet learn about the mappings you want? It would be possible (but tedious) to write a small Java class that simply instantiated new *ActionMapping* instances, and called all of the appropriate setter methods. To make this process easier, Struts uses the Jakarta-Digester component to parse an XML-based description of the desired mappings and create the appropriate objects initialized to the appropriate default values. See the [Jakarta Commons website](#) for more information about the Digester.

The developer's responsibility is to create an XML file named `struts-config.xml` and place it in the `WEB-INF` directory of your application. This format of this document is described by the Document Type Definition (DTD) maintained at [http://jakarta.apache.org/struts/dtds/struts-config\\_1\\_1.dtd](http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd). This chapter covers the configuration elements that you will typically write as part

of developing your application. There are several other elements that can be placed in the struts-config file to customize your application. See "[Configuring Applications](#)" for more about the other elements in the Struts configuration file.

The controller uses an internal copy of this document to parse the configuration; an Internet connection is not required for operation.

The outermost XML element must be `<struts-config>`. Inside of the `<struts-config>` element, there are three important elements that are used to describe your actions:

- `<form-beans>`
- `<global-forwards>`
- `<action-mappings>`

#### **`<form-beans>`**

This section contains your form bean definitions. Form beans are descriptors that are used to create `ActionForm` instances at runtime. You use a `<form-bean>` element for each form bean, which has the following important attributes:

- **name:** A unique identifier for this bean, which will be used to reference it in corresponding action mappings. Usually, this is also the name of the request or session attribute under which this form bean will be stored.
- **type:** The fully-qualified Java classname of the `ActionForm` subclass to use with this form bean.

#### **`<global-forwards>`**

This section contains your global forward definitions. Forwards are instances of the `ActionForward` class returned from an `ActionForm`'s `execute` method. These map logical names to specific resources (typically JSPs), allowing you to change the resource without changing references to it throughout your application. You use a `<forward>` element for each forward definition, which has the following important attributes:

- **name:** The logical name for this forward. This is used in your `ActionForm`'s `execute` method to forward to the next appropriate resource. Example: `homepage`
- **path:** The context relative path to the resource. Example: `/index.jsp` or `/index.do`
- **redirect:** `True` or `false` (default). Should the `ActionServlet` redirect to the resource instead of forward?

#### **`<action-mappings>`**

This section contains your action definitions. You use an `<action>` element for each of the mappings you would like to define. Most action elements will define at least the following attributes:

- **path:** The application context-relative path to the action.
- **type:** The fully qualified java classname of your `Action` class.
- **name:** The name of your `<form-bean>` element to use with this action

Other often-used attributes include:

- **parameter:** A general-purpose attribute often used by "standard" Actions to pass a required property.
- **roles:** A comma-delimited list of the user security roles that can access this mapping.

For a complete description of the elements that can be used with the `action` element, see the [Struts Configuration DTD](#) and the [ActionMapping documentation](#).

## 4.8.1 ActionMapping Example

The `struts-config.xml` file from the MailReader example application includes the following mapping entry for the "log on" function, which we will use to illustrate the requirements. Note that the entries for all the other actions are left out:

```
<struts-config>
  <form-beans>
    <form-bean
      name="logonForm"
      type="org.apache.struts.webapp.example.LogonForm" />
    </form-beans>
  <global-forwards>
    <forward
      type="org.apache.struts.action.ActionForward" />
    <forward
      name="logon"
      path="/logon.jsp"
      redirect="false" />
    </global-forwards>
  <action-mappings>
    <action
      path="/logon"
      type="org.apache.struts.webapp.example.LogonAction"
      name="logonForm"
      scope="request"
      input="/logon.jsp"
      unknown="false"
      validate="true" />
    </action-mappings>
</struts-config>
```

First the form bean is defined. A basic bean of class `"org.apache.struts.webapp.example.LogonForm"` is mapped to the logical name `"logonForm"`. This name is used as a session or request attribute name for the form bean.

The `"global-forwards"` section is used to create logical name mappings for commonly used presentation pages. Each of these forwards is available through a call to your action mapping instance, i.e. `mapping.findForward("logicalName")`.

As you can see, this mapping matches the path `/logon` (actually, because the MailReader example application uses extension mapping, the request URI you specify in a JSP page would end in `/logon.do`). When a request that matches this path is received, an instance of the *LogonAction* class will be created (the first time only) and used. The controller servlet will look for a session scoped bean under key `logonForm`, creating and saving a bean of the specified class if needed.

Optional but very useful are the local `"forward"` elements. In the MailReader example application, many actions include a local `"success"` and/or `"failure"` forward as part of an action mapping.

```
<!-- Edit mail subscription -->
<action
  path="/editSubscription"
  type="org.apache.struts.webapp.example.EditSubscriptionAction"
```

```

        name="subscriptionForm"
        scope="request"
        validate="false">
        <forward
            name="failure"
            path="/mainMenu.jsp"/>
        <forward
            name="success"
            path="/subscription.jsp"/>
    </action>

```

Using just these two extra properties, the Action classes are almost totally independent of the actual names of the presentation pages. The pages can be renamed (for example) during a redesign, with negligible impact on the Action classes themselves. If the names of the "next" pages were hard coded into the Action classes, all of these classes would also need to be modified. Of course, you can define whatever local `forward` properties makes sense for your own application.

The Struts configuration file includes several other elements that you can use to customize your application. See "[Configuring Applications](#)" for details.

## 4.9 Using ActionMappings for Pages

Fronting your pages with ActionMappings is *essential* when using application modules, since doing so is the only way you involve the controller in the request -- and you want to! The controller puts the application configuration in the request, which makes available all of your module-specific configuration data (including which message resources you are using, request-processor, datasources, and so forth).

The simplest way to do this is to use the `forward` property of the ActionMapping:

```
<action path="/view" forward="/view.jsp"/>
```

## 4.10 Commons Logging Interface

Struts doesn't configure logging itself -- it's all done by [commons-logging](#) under the covers. The default algorithm is a search:

- If Log4J is there, use it.
- If JDK 1.4 is there, use it.
- Otherwise, use SimpleLog.

The commons-logging interface is an *ultra-thin* bridge to many different logging implementations. The intent is to remove compile- and run-time dependencies on any single logging implementation. For more information about the currently-supported implementations, please refer to the [the description for the org.apache.commons.logging package](#).

Because Struts uses commons-logging and, therefore, includes the necessary JAR files for **you** to use commons-logging, you've probably had the occasional fleeting thought, "*Should I use commons-logging?*" The answer (surprise!) depends on the requirements for your particular project. If one of your requirements is the ability to easily change logging implementations with zero impact on your application, then commons-logging is a very good option.

*"Great! What do I do to get started using commons-logging in my own code?"*

Using commons-logging in your own code is very simple - all you need are two imports and a declaration for a logger. Let's take a look:

```
package com.foo;
// ...
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
...
public class Foo {
    // ...
    private static Log log = LogFactory.getLog(Foo.class);
    // ...
    public void setBar(Bar bar) {
        if (log.isTraceEnabled()) {
            log.trace("Setting bar to " + bar);
        }
        this.bar = bar;
    }
// ...
}
```

The general idea is to instantiate a single logger per class and to use a name for the logger which reflects where it's being used. The example is constructed with the class itself. This gives the logger the name of `com.foo.Foo`. Doing things this way lets you easily see where the output is coming from, so you can quickly pin-point problem areas. In addition, you are able to enable/disable logging in a very fine-grained way.

For examples of using logging in Struts classes, see the Action classes in the Struts MailReader example application.

Next: [Configuring Applications](#)



## User Guide

[Table of Contents](#)  
[Preface](#)  
[Introduction](#)  
[Model Components](#)  
[View Components](#)  
[Controller Components](#)  
[Configuration](#)  
[Release Notes](#)  
[Installation](#)

## Developer Guides

[Bean Tags](#)  
[HTML Tags](#)  
[Logic Tags](#)  
[Nested Tags](#)  
[Template Tags](#)  
[Tiles Tags](#)  
[Utilities](#)  
[Validator](#)

## Quick Links

[Welcome](#)  
[News and Status](#)  
[Resources](#)  
[User and Developer Guides \\*](#)  
[FAQs and HowTos](#)

## 5. Configuring Applications

Contributors:

- Craig R. McClanahan
- Mike Schachter
- Ted Husted
- Martin Cooper
- Ed Burns
- Donald Ball
- Eddie Bush
- Yann Cebron
- David Graham
- Tim O'Brien

### 5.1 Overview

Before you can build an application, you need to lay a solid foundation. There are several setup tasks you need to perform before deploying your Struts application. These include components in the Struts configuration file and in the Web Application Deployment Descriptor.

### 5.2 The Struts configuration file

The [Building Controller Components](#) chapter covered writing the form-bean and action-mapping portions of the Struts configuration file. These elements usually play an important role in the development of a Struts application. The other elements in Struts configuration file tend to be static: you set them once and leave them alone.

These "static" configuration elements are:

- controller
- message-resources
- plug-in
- data-sources

#### 5.2.1 Controller Configuration

The `<controller>` element allows you to configure the `ActionServlet`. Many of the controller parameters were previously defined by servlet initialization parameters in your `web.xml` file but have been moved to this section of `struts-config.xml` in order to allow different modules in the same web application to be configured differently. For full details on available parameters see the [struts-config\\_1\\_1.dtd](#) or the list below.

- **bufferSize** - The size (in bytes) of the input buffer used when processing file uploads. [4096] (optional)
- **className** - Classname of configuration bean.  
[org.apache.struts.config.ControllerConfig] (optional)
- **contentType** - Default content type (and optional character encoding) to be set on each response. May be overridden by the Action, JSP, or other resource to which the request is forwarded. [text/html] (optional)
- **forwardPattern** - Replacement pattern defining how the "path" attribute of a `<forward>` element is mapped to a context-relative URL when it starts with a slash (and when the `contextRelative` property is `false`). This value may consist of any combination of the following:
  - `$M` - Replaced by the module prefix of this module.
  - `$P` - Replaced by the "path" attribute of the selected `<forward>` element.
  - `$$` - Causes a literal dollar sign to be rendered.
  - `$x` - (Where "x" is any character not defined above) Silently swallowed, reserved for future use.

If not specified, the default `forwardPattern` is consistent with the previous behavior of forwards. [`$M$P`] (optional)

- **inputForward** - Set to `true` if you want the `input` attribute of `<action>` elements to be the name of a local or global `ActionForward`, which will then be used to calculate the ultimate URL. Set to `false` to treat the `input` parameter of `<action>` elements as a module-relative path to the resource to be used as the input form. [`false`] (optional)
- **locale** - Set to `true` if you want a `Locale` object stored in the user's session if not already present. [`true`] (optional)
- **maxFileSize** - The maximum size (in bytes) of a file to be accepted as a file upload. Can be expressed as a number followed by a "K", "M", or "G", which are interpreted to mean kilobytes, megabytes, or gigabytes, respectively. [250M] (optional)
- **multipartClass** - The fully qualified Java class name of the multipart request handler class to be used with this module.  
[org.apache.struts.upload.CommonsMultipartRequestHandler] (optional)
- **nocache** - Set to `true` if you want the controller to add HTTP headers for defeating caching to every response from this module. [`false`] (optional)
- **pagePattern** - Replacement pattern defining how the `page` attribute of custom tags using it is mapped to a context-relative URL of the corresponding resource. This value may consist of any combination of the following:
  - `$M` - Replaced by the module prefix of this module.
  - `$P` - Replaced by the "path" attribute of the selected `<forward>` element.
  - `$$` - Causes a literal dollar sign to be rendered.
  - `$x` - (Where "x" is any character not defined above) Silently swallowed, reserved for future use.

If not specified, the default `pagePattern` is consistent with the previous behavior of URL calculation. [`$M$P`] (optional)

- **processorClass** - The fully qualified Java class name of the `RequestProcessor` subclass to be used with this module.  
[org.apache.struts.action.RequestProcessor] (optional)



- **tempDir** - Temporary working directory to use when processing file uploads. [{the directory provided by the servlet container}]

This example uses the default values for several controller parameters. If you only want default behavior you can omit the controller section altogether.

```
<controller
  processorClass="org.apache.struts.action.RequestProcessor"
  debug="0"
  contentType="text/html" />;
```

## 5.2.2 Message Resources Configuration

Struts has built in support for internationalization (I18N). You can define one or more `<message-resources>` elements for your webapp; modules can define their own resource bundles. Different bundles can be used simultaneously in your application, the 'key' attribute is used to specify the desired bundle.

- **className** - Classname of configuration bean.  
[org.apache.struts.config.MessageResourcesConfig] (optional)
- **factory** - Classname of MessageResourcesFactory.  
[org.apache.struts.util.PropertyMessageResourcesFactory] (optional)
- **key** - ServletContext attribute key to store this bundle.  
[org.apache.struts.action.MESSAGE] (optional)
- **null** - Set to false to display missing resource keys in your application like '???keyname???' instead of null. [true] (optional)
- **parameter** - Name of the resource bundle. (required)

Example configuration:

```
<message-resources
  parameter="MyWebAppResources"
  null="false" />
```

This would set up a message resource bundle provided in the file `MyWebAppResources.properties` under the default key. Missing resource keys would be displayed as `???keyname???`.

## 5.2.3 PlugIn Configuration

Struts PlugIns are configured using the `<plug-in>` element within the Struts configuration file. This element has only one valid attribute, 'className', which is the fully qualified name of the Java class which implements the `org.apache.struts.action.PlugIn` interface.

For PlugIns that require configuration themselves, the nested `<set-property>` element is available.

This is an example using the Tiles plugin:

```
<plug-in className="org.apache.struts.tiles.TilesPlugin" >
  <set-property
    property="definitions-config"
    value="/WEB-INF/tiles-defs.xml" />
```

```
</plug-in>
```

## 5.2.4 Data Source Configuration

Besides the objects related to defining ActionMappings, the Struts configuration may contain elements that create other useful objects.

The `<data-sources>` section can be used to specify a collection of DataSources [javax.sql.DataSource] for the use of your application. Typically, a DataSource represents a connection pool to a database or other persistent store. As a convenience, the Struts DataSource manager can be used to instantiate whatever standard pool your application may need. Of course, if your persistence layer provides for its own connections, then you do not need to specify a `data-sources` element.

Since DataSource implementations vary in what properties need to be set, unlike other Struts configuration elements, the `data-source` element does not pre-define a slate of properties. Instead, the generic `set-property` feature is used to set whatever properties your implementation may require. Typically, these settings would include:

- A driver class name
- A url to access the driver
- A description

And other sundry properties.

As a further convenience, Struts provides a default DataSource implementation, [org.apache.struts.util.GenericDataSource]. The `type` property can be used to specify another implementation:

```
<data-source type="org.apache.commons.dbcp.BasicDataSource">
<!-- ... set-property elements ... -->
</data-source>
```

In Struts 1.1, the GenericDataSource is deprecated, and it is recommended that you use the Commons BasicDataSource directly. In practice, if you need to use the DataSource manager, you should use whatever DataSource implementation works best with your container or database.

For examples of specifying a data-sources element and using the DataSource with an Action, see the [Accessing a Database HowTo](#).

## 5.3 Configuring your application for modules

Very little is required in order to start taking advantage of the Struts application module feature. Just go through the following steps:

1. Prepare a config file for each module.
2. Inform the controller of your module.
3. Use actions to refer to your pages.

### 5.3.1 Module Configuration Files

Back in Struts 1.0, a few "boot-strap" options were placed in the web.xml file, and the bulk of the configuration was done in a single struts-config.xml file. Obviously, this wasn't ideal for a team environment, since multiple users had to share the same configuration file.

In Struts 1.1, you have two options: you can list [multiple struts-config files](#) as a comma-delimited list, or you can subdivide a larger application into modules.

With the advent of modules, a given module has its own configuration file. This means each team (each module would presumably be developed by a single team) has their own configuration file, and there should be a lot less contention when trying to modify it.

## 5.3.2 Informing the Controller

In struts 1.0, you listed your configuration file as an initialization parameter to the action servlet in web.xml. This is still done in 1.1, but it's augmented a little. In order to tell the Struts machinery about your different modules, you specify multiple config initialization parameters, with a slight twist. You'll still use "config" to tell the action servlet about your "default" module, however, for each additional module, you will list an initialization parameter named "config/module", where module is the name of your module (this gets used when determining which URIs fall under a given module, so choose something meaningful!). For example:

```
...
<init-param>
  <param-name>config</param-name>
  <param-value>/WEB-INF/conf/struts-default.xml</param-value>
</init-param>
<init-param>
  <param-name>config/module1</param-name>
  <param-value>/WEB-INF/conf/struts-module1.xml</param-value>
</init-param>
...
```

This says I have two modules. One happens to be the "default" module, which has no "/module" in it's name, and one named "module1" (config/module1). I've told the controller it can find their respective configurations under /WEB-INF/conf (which is where I put all my configuration files). Pretty simple!

(My struts-default.xml would be equivalent to what most folks call struts-config.xml. I just like the symmetry of having all my Struts module files being named struts-<module>.xml)

## 5.3.3 Switching Modules

There are two basic methods to switching from one module to another. You can either use a forward (global or local) and specify the `contextRelative` attribute with a value of `true`, or you can use the built-in `org.apache.struts.actions.SwitchAction`.

Here's an example of a global forward:

```
...
<struts-config>
...
<global-forwards>
<forward name="toModuleB"
  contextRelative="true"
  path="/moduleB/index.do"
  redirect="true"/>
...
</global-forwards>
...
</struts-config>
```

You could do the same thing with a local forward declared in an `ActionMapping`:

```
...
<struts-config>
...
<action-mappings>
...
<action ... >
<forward name="success"
  contextRelative="true"
  path="/moduleB/index.do"
  redirect="true"/>
</action>
...
</action-mappings>
...
</struts-config>
```

Finally, you could use `org.apache.struts.actions.SwitchAction`, like so:

```
...
<action-mappings>
<action path="/toModule"
  type="org.apache.struts.actions.SwitchAction"/>
...
</action-mappings>
...
```

Now, to change to ModuleB, we would use a URI like this:

```
http://localhost:8080/toModule.do?prefix=moduleB&page=index.do
```

That's all there is to it! Happy module-switching!

## 5.4 The Web Application Deployment Descriptor

The final step in setting up the application is to configure the application deployment descriptor (stored in file `WEB-INF/web.xml`) to include all the Struts components that are required. Using the deployment descriptor for the example application as a guide, we see that the following entries need to be created or modified.

### 5.4.1 Configure the Action Servlet Instance

Add an entry defining the action servlet itself, along with the appropriate initialization parameters. Such an entry might look like this:

```
<servlet>
  <servlet-name>action</servlet-name>
  <servlet-class>
    org.apache.struts.action.ActionServlet
  </servlet-class>
  <init-param>
    <param-name>application</param-name>
    <param-value>
      org.apache.struts.webapp.example.ApplicationResources
    </param-value>
  </init-param>
  <init-param>
    <param-name>config</param-name>
    <param-value>
      /WEB-INF/struts-config.xml
    </param-value>
  </init-param>
  <init-param>
    <param-name>debug</param-name>
    <param-value>2</param-value>
  </init-param>
  <init-param>
    <param-name>mapping</param-name>
    <param-value>
      org.apache.struts.webapp.example.ApplicationMapping
    </param-value>
  </init-param>
  <load-on-startup>2</load-on-startup>
</servlet>
```

The initialization parameters supported by the controller servlet are described below. (You can also find these details in the [Javadocs](#) for the `ActionServlet` class.) Square brackets describe the default values that are assumed if you do not provide a value for that initialization parameter.

- **config** - Context-relative path to the XML resource containing the configuration information for the default module. This may also be a comma-delimited list of configuration files. Each file is loaded in turn, and its objects are appended to the internal data structure. `[/WEB-INF/struts-config.xml]`.  
**WARNING** - If you define an object of the same name in more than one configuration file, the last one loaded quietly wins.
- **config/{module}** - Context-relative path to the XML resource containing the configuration information for the application module that will use the specified prefix

(/{module}). This can be repeated as many times as required for multiple application modules. (Since Struts 1.1)

- **convertNull** - Force simulation of the Struts 1.0 behavior when populating forms. If set to true, the numeric Java wrapper class types (like `java.lang.Integer`) will default to null (rather than 0). (Since Struts 1.1) [false]
- **debug** - The debugging detail level that controls how much information is logged for this servlet. Accepts values 0 (off) and from 1 (least serious) through 6 (most serious). [0]
- **detail** - The debugging detail level for the Digester we utilize to process the application module configuration files. Accepts values 0 (off) and 1 (least serious) through 6 (most serious). [0]
- **rulesets** - Comma-delimited list of fully qualified classnames of additional `org.apache.commons.digester.RuleSet` instances that should be added to the Digester that will be processing `struts-config.xml` files. By default, only the RuleSet for the standard configuration elements is loaded. (Since Struts 1.1)
- **validating** - Should we use a validating XML parser to process the configuration file (strongly recommended)? [true]

The following parameters may still be used with the Struts 1.1 release but are **deprecated**.

- **application** - Java class name of the application resources bundle base class. [NONE]  
*DEPRECATED - Configure this using the "parameter" attribute of the <message-resources> element.*
- **bufferSize** - The size of the input buffer used when processing file uploads. [4096]  
*DEPRECATED - Configure this using the "bufferSize" attribute of the <controller> element.*
- **content** - Default content type and character encoding to be set on each response; may be overridden by a forwarded-to servlet or JSP page. [text/html] *DEPRECATED - Configure this using the "contentType" attribute of the <controller> element.*
- **factory** - The Java class name of the MessageResourcesFactory used to create the application MessageResources object.  
[org.apache.struts.util.PropertyMessageResourcesFactory] *DEPRECATED - Configure this using the "factory" attribute of the <message-resources> element.*
- **formBean** - The Java class name of the ActionFormBean implementation to use  
[org.apache.struts.action.ActionFormBean]. *DEPRECATED - Configure this using the "className" attribute of each <form-bean> element.*
- **forward** - The Java class name of the ActionForward implementation to use  
[org.apache.struts.action.ActionForward]. Two convenient classes you may wish to use are:
  - `org.apache.struts.action.ForwardingActionForward` - Subclass of `org.apache.struts.action.ActionForward` that defaults the `redirect` property to false (same as the ActionForward default value).
  - `org.apache.struts.action.RedirectingActionForward` - Subclass of `org.apache.struts.action.ActionForward` that defaults the `redirect` property to true.*DEPRECATED - Configure this using the "className" attribute of each <forward> element.*
- **locale** - If set to true, and there is a user session, identify and store an appropriate `java.util.Locale` object (under the standard key identified by `Action.LOCALE_KEY`) in the user's session if there is not a Locale object there already.  
[true] *DEPRECATED - Configure this using the "locale" attribute of the <controller> element.*
- **mapping** - The Java class name of the ActionMapping implementation to use  
[org.apache.struts.action.ActionMapping]. Two convenient classes you may wish to use are:
  - `org.apache.struts.action.RequestActionMapping` - Subclass of `org.apache.struts.action.ActionMapping` that defaults the `scope` property to "request".

- *org.apache.struts.action.SessionActionMapping* - Subclass of *org.apache.struts.action.ActionMapping* that defaults the scope property to "session". (Same as the *ActionMapping* default value).  
*DEPRECATED* - Configure this using the "className" attribute of each <action> element, or globally for a module by using the "type" attribute of the <action-mappings> element.
- **maxFileSize** - The maximum size (in bytes) of a file to be accepted as a file upload. Can be expressed as a number followed by a "K" "M", or "G", which are interpreted to mean kilobytes, megabytes, or gigabytes, respectively. [250M] *DEPRECATED* - Configure this using the "maxFileSize" attribute of the <controller> element.
- **multipartClass** - The fully qualified name of the *MultipartRequestHandler* implementation class to be used for processing file uploads. If set to none, disables Struts multipart request handling. [org.apache.struts.upload.CommonsMultipartRequestHandler] *DEPRECATED* - Configure this using the "multipartClass" attribute of the <controller> element.
- **nocache** - If set to true, add HTTP headers to every response intended to defeat browser caching of any response we generate or forward to. [false] *DEPRECATED* - Configure this using the "nocache" attribute of the <controller> element.
- **null** - If set to true, set our application resources to return null if an unknown message key is used. Otherwise, an error message including the offending message key will be returned. [true] *DEPRECATED* - Configure this using the "null" attribute of the <message-resources> element.
- **tempDir** - The temporary working directory to use when processing file uploads. [The working directory provided to this web application as a servlet context attribute]  
*DEPRECATED* - Configure this using the "tempDir" attribute of the <controller> element.

## 5.4.2 Configure the Action Servlet Mapping

**Note:** The material in this section is not specific to Struts. The configuration of servlet mappings is defined in the Java Servlet Specification. This section describes the most common means of configuring a Struts application.

There are two common approaches to defining the URLs that will be processed by the controller servlet -- prefix matching and extension matching. An appropriate mapping entry for each approach will be described below.

Prefix matching means that you want all URLs that start (after the context path part) with a particular value to be passed to this servlet. Such an entry might look like this:

```
<servlet-mapping>
  <servlet-name>action</servlet-name>
  <url-pattern>/do/*</url-pattern>
</servlet-mapping>
```

which means that a request URI to match the /logon path described earlier might look like this:

```
http://www.mycompany.com/myapplication/do/logon
```

where /myapplication is the context path under which your application is deployed.

Extension mapping, on the other hand, matches request URIs to the action servlet based on the fact that the URI ends with a period followed by a defined set of characters. For example, the JSP processing servlet is mapped to the \*.jsp pattern so that it is called to process every JSP page that is requested. To use the \*.do extension (which implies "do something"), the mapping entry would look like this:



```

<servlet-mapping>
  <servlet-name>action</servlet-name>
  <url-pattern>*.do</url-pattern>
</servlet-mapping>

```

and a request URI to match the /logon path described earlier might look like this:

```
http://www.mycompany.com/myapplication/logon.do
```

**WARNING** - Struts will not operate correctly if you define more than one <servlet-mapping> element for the controller servlet.

**WARNING** - If you are using the new module support in Struts 1.1, you should be aware that **only** extension mapping is supported.

### 5.4.3 Configure the Struts Tag Library

Next, you must add an entry defining the Struts tag library. There are six taglibs included with the Struts distribution.

The struts-bean taglib contains tags useful in accessing beans and their properties, as well as defining new beans (based on these accesses) that are accessible to the remainder of the page via scripting variables and page scope attributes. Convenient mechanisms to create new beans based on the value of request cookies, headers, and parameters are also provided.

The struts-html taglib contains tags used to create struts input forms, as well as other tags generally useful in the creation of HTML-based user interfaces.

The struts-logic taglib contains tags that are useful in managing conditional generation of output text, looping over object collections for repetitive generation of output text, and application flow management.

The struts-template taglib contains tags that define a template mechanism.

The struts-tiles taglib contains tags used for combining various view components, called "tiles", into a final composite view. This is similar to struts-template in that it's used for view composition, but tiles is a more full featured set of tags.

The struts-nested taglib is an extension of other struts taglibs that allows the use of nested beans.

Below is how you would define all taglibs for use within your application. In practice, you would only specify the taglibs that your application uses:

```

<taglib>
  <taglib-uri>
    /tags/struts-bean
  </taglib-uri>
  <taglib-location>
    /WEB-INF/struts-bean.tld
  </taglib-location>
</taglib>
<taglib>
  <taglib-uri>

```

```

        /tags/struts-html
    </taglib-uri>
    <taglib-location>
        /WEB-INF/struts-html.tld
    </taglib-location>
</taglib>
<taglib>
    <taglib-uri>
        /tags/struts-logic
    </taglib-uri>
    <taglib-location>
        /WEB-INF/struts-logic.tld
    </taglib-location>
</taglib>
<taglib>
    <taglib-uri>
        /tags/struts-tiles
    </taglib-uri>
    <taglib-location>
        /WEB-INF/struts-tiles.tld
    </taglib-location>
</taglib>

```

This tells the JSP system where to find the tag library descriptor for this library (in your application's WEB-INF directory, instead of out on the Internet somewhere).

#### 5.4.3.1 Configure the Struts Tag Library (Servlet 2.3)

**Servlet 2.3 Users only:** The Servlet 2.3 specification simplifies the deployment and configuration of tag libraries. The instructions above will work on older containers as well as 2.3 containers (Struts only requires a servlet 2.2 container); however, if you're using a 2.3 container such as Tomcat 4.x, you can take advantage of a simplified deployment.

All that's required to install the struts tag libraries is to copy struts.jar into your /WEB-INF/lib directory and reference the tags in your code like this:

```

<%@ taglib
    uri="http://jakarta.apache.org/struts/tags-html-1.0"
    prefix="html" %>

```

Note that you **must use the full uri** defined in the various struts tlds so that the container knows where to find the tag's class files. You don't have to alter your web.xml file or copy tlds into any application directories.

### 5.5 Add Struts Components To Your Application

To use Struts, you must copy the .tld files that you require into your WEB-INF directory, and copy struts.jar (and all of the commons-\*.jar files) into your WEB-INF/lib directory.

**Servlet 2.3 Users:** See [section 4.5.3.1](#) for how to avoid copying the tlds into your application.

For more about putting it all together, see the [Building Applications HowTo](#).

Next: [Release Notes](#)

---

*Copyright (c) 2000-2002, Apache Software Foundation*

**Powered by  
Struts**



## User Guide

[Table of Contents](#)[Preface](#)[Introduction](#)[Model](#)[Components](#)[View](#)[Components](#)[Controller](#)[Components](#)[Configuration](#)[Release Notes](#)[Installation](#)

## Developer Guides

[Bean Tags](#)[HTML Tags](#)[Logic Tags](#)[Nested Tags](#)[Template Tags](#)[Tiles Tags](#)[Utilities](#)[Validator](#)

## Quick Links

[Welcome](#)[News and Status](#)[Resources](#)[User and  
Developer  
Guides \\*](#)[FAQs and  
HowTos](#)

## 6.1 Release Notes

Contributors:

- Craig R. McClanahan
- Robert Leland
- Ted Husted
- Martin Cooper

## Beta Notes

This section contains the release notes for **nightly build** of the Struts Framework, for changes that have taken place since [Version 1.1 beta 2](#) was released. For a complete list of changes since the last production release, see the [Introduction](#)

## Beta Fixes

**ApplicationConfig:** In Struts 1.1. beta 3, the ApplicationConfig class is renamed to ModuleConfig, to conform with the venacular.

**Dyna\*Form:** In Struts 1.1. beta 3, the reset method was changed so that it conforms to the original ActionForm implementation.

**Blank application:** Fixed configuration problem with Struts Blank application.

## Introduction

The remainder of this document contains the release notes for **nightly build** of the Struts Framework, and covers changes that have taken place since [Version 1.0.2](#) was released. The following sections cover [New Features](#) and [Changes](#) to Struts.

## What's Included?

The binary distribution of this release includes the following files relevant to Struts:

- **INSTALL** - Brief installation instructions. See the [Struts Documentation Application](#), or online at <http://jakarta.apache.org/struts/> for more information.
- **LICENSE** - The Apache Software Foundation license that defines the terms under which you can use Struts (and other software licensed by Apache).
- **README** - A brief introduction to Struts.
- **lib/** - Directory containing files you will need in your own applications. The individual files of interest are:
  - **commons-\*.jar** - Release packages from the [Jakarta Commons Project](#) that Struts relies on. You are welcome to use these classes in your own applications. These JAR files should be copied into the `/WEB-INF/lib` directory of your web application.
  - **struts.jar** - JAR file that contains the compiled Java classes of Struts. You must place this file in the `/WEB-INF/lib` directory of your web application.
  - **struts-xxxxx.tld** - The tag library descriptor files for the Struts 1.1 tag libraries (bean, html, logic, and template). You must place these files in the `/WEB-INF` directory of your web application, and reference them with appropriate `<taglib>` directives in your web.xml file.
  - **jdbc2\_0-stdext.jar** - The JDBC 2.0 Optional Package API classes (package `javax.sql`). You will need to include this file in the `/WEB-INF/lib` directory of your application, if it is not already made visible to web applications by your servlet container.
  - **struts-config\_1\_1.dtd** - The document type descriptor (DTD) for the Struts 1.1 configuration file (which is typically named `/WEB-INF/struts-config.xml`). Your configuration file will be validated against an internal copy of this DTD -- this copy is available for reference purposes only.
  - **struts-config\_1\_0.dtd** - The document type descriptor (DTD) for the Struts 1.0 configuration file (which is typically named `/WEB-INF/struts-config.xml`). Your configuration file will be validated against an internal copy of this DTD -- this copy is available for reference purposes only.
  - **web-app\_2\_2.dtd** - The document type descriptor (DTD) for web.xml files conforming to the Servlet 2.2 specification. This copy is for reference purposes only.
  - **web-app\_2\_3.dtd** - The document type descriptor (DTD) for web.xml files conforming to the Servlet 2.3 specification. This copy is for reference purposes only.
- **webapps/** - Web Application Archive (WAR) files for the web applications that are included with Struts.

## What's New?

Following are highlights of the new features. In the next section, we provide links to the JavaDocs for the affected classes.

### New Configuration DTD

The Struts Configuration 1.0 DTD has been deprecated in favor of the [struts-config\\_1\\_1.dtd](#). In the Struts 1.1 release, existing Struts configuration files can be loaded using either DTD version.

### New Dependencies on Commons packages

Several components of Struts 1.0 have been found to be useful in general Java development (and not just useful for building Struts-based web applications), and have been migrated into the [Jakarta Commons Project](#). As a result, the current development version of Struts has been modified to rely on the Commons packages containing these classes, rather than the Struts internal versions. In nearly every case, this involved changing only the `import` statements at the top of your classes. Any applications that utilize these classes will need to be modified in the same way.

The following Commons packages contain the replacements for the corresponding Struts 1.0 classes:

- **BeanUtils Package** [[org.apache.commons.beanutils](#)] -  
`org.apache.struts.utils.BeanUtils,`  
`org.apache.struts.utils.ConvertUtils,` and  
`org.apache.struts.utils.PropertyUtils.`
- **Collections Package** [[org.apache.commons.collections](#)] -  
`org.apache.struts.util.ArrayStack,`  
`org.apache.struts.util.FastArrayList,`  
`org.apache.struts.util.FastHashMap,`  
`org.apache.struts.util.FastTreeMap.`
- **Digester Package** - [[org.apache.commons.digester](#)] -  
`org.apache.struts.digester.*.`

The following Commons packages are also now used by various components of the Struts framework:

- **Database Connection Pool Package** [[org.apache.commons.dbpc](#)]
- **FileUpload Package** [[org.apache.commons.fileupload](#)]
- **Logging Package** [[org.apache.commons.logging](#)]
- **Pool Package** [[org.apache.commons.pool](#)]
- **Validator Package** [[org.apache.commons.validator](#)]

### **NOTE! XML Parser Prerequisite Updated**

Struts now depends on an XML parser that conforms to the JAXP/1.1 (rather than JAXP/1.0) APIs. Parsers known to work include the JAXP/1.1 reference implementation, and Xerces 1.3.1.

### **SOURCE DEVELOPERS NOTE! Ant Prerequisite Updated**

To build Struts from source Ant 1.4 or later is now required. This does not affect developers that use Struts from the binary distribution.

### **Struts Validator Integration**

The new Commons-Validator is now integrated with Struts and exposed through the new Validator package.

### **Tiles - An advanced templating taglib**

The Tiles JSP assembly framework has been integrated with Struts.

### **Nested - An very cool taglib extension**

The Nested taglib is bundled with Struts and enhances the functionality of the existing Struts

tags.

## **New Example Applications**

New example applications for the Validator and Tiles are now part of the Struts distribution.

## **New Contrib directory for optional components**

A new directory (`contrib`) in the CVS source repository has been added to accumulate Struts add-on extensions that are generally useful but have not yet been integrated into the standard code base.

- Scaffold - An extension of the Commons Scaffold toolkit of reusable classes for building web applications.
- Struts-el - The optional Struts-el taglib makes it easy to use Struts with JSTL (container with servlet 2.3 support required).

The source for these components is available in the Struts source distribution. Binary distributions may also be made available with the Struts download area. As optional components, these products have their own release cycles.

## **Action Package Additions**

The following new features have been added to the basic controller framework [`org.apache.struts.action`]:

- The `ActionServlet` now provides support for modular Struts applications and sports several new extension points.
- The new `ActionMessages` class will support a superset of the capabilities of `ActionErrors`, and will be useful as a collection of general purpose messages, not just errors.

## **Upload Package Additions**

The following new features have been added to the file upload classes [`org.apache.struts.upload`]:

- `CommonsMultipartRequestHandler`: New class that implements file upload using the Jakarta Commons FileUpload package.

## **Util Package Additions**

The following new features have been added to the utility classes [`org.apache.struts.util`]:

- `LocalStrings`: Correct message regarding replaceable parameter so that it does not append an extraneous character.
- `LabelValueBean`: New class that defines a collection of name/value pairs that can be used with the `<html:options>` tag, and elsewhere.
- `MessageResources`: Escape any single quote characters that are included in the specified message string.
- `computeParameters`: Allow a transaction token to be the only parameter in .
- `RequestUtils`: Change to encode ampersands when building a query string.

## **Bean Taglib Package Additions**



The following new features have been added to the *struts-bean* custom tag library [org.apache.struts.taglib.bean]:

- `<bean:write>`: Add format, locale and bundle attributes to support formatting values according to current user locale, format string from attribute or format string from string resources.
- `<bean:cookie>`, `<bean:header>`, or `<bean:parameter>`: Correct the generated scripting variable type when tag is used with the "multiple" attribute.
- `<bean:message>`: Added name, property, and scope attributes to the tag, so that the message source key can be obtained dynamically from a bean or bean property.

### HTML Taglib Package Additions

The following new features have been added to the *struts-html* custom tag library [org.apache.struts.taglib.html]:

- `<html:link>`: Added 'action' attribute.
- `<html:options>`: If the property specified by the 'property' attribute returns null, tag now throws an error message that indicates what the real problem is, rather than causing an NPE.
- `<html:option>` and `<html:options>`: Added 'style' and 'styleClass' attributes.
- `<html:optionsCollection>`: New tag providing a cleaner way of populating HTML options from a collection.
- `<bean:message>`: Added 'name', 'property' and 'scope' attributes so that the message resource key can be obtained dynamically from a bean.
- `<html:messages>`: New tag to iterate through a message collection in the new ActionMessages class.
- `ActionForm`: Tag will now call `reset()` if it instantiates the ActionForm bean. This also requires that the bean instantiated by the tag to be an ActionForm subclass.
- `<html:image>`: Added the 'align' attribute.
- `<html:img>`: Added the mouse event attributes ('onclick', 'ondblclick', 'onmousedown', 'onmouseup', 'onmouseover', 'onmousemove', 'onmouseout').
- `SubmitTag`, `SelectTag`, `LinkTag.java`, `CheckboxTag`, `ButtonTag`, `ImageTag`, `RadioTag`, and `TextArea` tags: Added indexed property.

### Logic Taglib Package Additions

The following new features have been added to the *struts-logic* custom tag library [org.apache.struts.taglib.logic]:

- `<logic:empty>` and `<logic:notEmpty>`: New tags that are similar to `<logic:present>` and `<logic:notPresent>` except for the treatment of empty strings.

### Template Taglib Package Additions

The following new features have been added to the *struts-template* custom tag library [org.apache.struts.taglib.template]:

- None.

### Documentation Additions

The following new features have been added to the Struts Documentation application (and corresponding contents on the Struts web site):

- Version Differences: New section in Release Notes to link to the JavaDocs for all Struts classes and members added or changed between versions.
- FAQ/HowTos: New documentation category organizes our FAQs and example-driven howTos. New HowTos include "Building Applications", "Using SSL", and using Struts with Eclipse or NetBeans.
- User Guide Preface: New section to overview the enabling technologies behind Struts.
- Developer Guides: Added "cover page" to guides. These then link to the Package Descriptions and the API guides.
- HTML tag documentation: expanded to cover using indexed properties with iterate.
- Site Menu: Removed separate links to taglib documentation, since these are now in the Developer Guide.
- Newbie FAQ: The questions most likely to be asked by new developers using Struts. Still under development.
- Kickstart FAQ: The questions most likely to be asked when selecting Struts.
- 1.0 JavaDoc: Added archival copy to web site for future reference.
- The UserGuide "Building" pages: General revisions to reflect new features and current practices.
- Installation: Updated instructions for SilverStream and Resin. Add installation notes for Jetty. Added RexIP to list of nominal containers.
- JavaDocs: New `@since Struts 1.1` tag to indicate new packages, classes, and members added after the Struts 1.0.x version

## Operational Changes and Bug Fixes

### Struts Configuration Changes

The following changes and bug fixes have occurred in the configuration files related to Struts:

- Deprecated (Struts 0.5) configuration file format: Remove support.
- Deprecated (Struts 0.5) methods: Remove from codebase.

### Added Config Package

- ControllerConfig: Added `inputForward` property to indicate that `ActionMapping.input` is a forward rather than a URI.
- ControllerConfig: Added `forwardPattern` and `inputPattern` to help manage page directories in application modules.
- Added package to provide more flexibility in configuring the controller and to provide support for modular applications

### Action Package Changes

The following changes and bug fixes have occurred in the basic controller framework (package `org.apache.struts.action`):

- ActionMapping: `input` property may now refer to an `ActionForward` rather than a module-relative path if `inputForward` is set to true on the module's `ControllerConfig` bean [`org.apache.struts.config.ControllerConfig`].
- ActionServlet: Added `convertNull` parameter to simulate the Struts 1.0 behavior when populating forms. If set to true, the numeric Java wrapper class types (like `java.lang.Integer`) will default to null (rather than 0).

- **ActionServlet:** Added "config/\$foo" parameter and deprecated several others in favor of components in the new config package.
- **ActionForms and related classes:** now use a StringBuffer when responding a toString request in order to conserve resources.
- **LookupDispatchAction:** Added standard Action to help select between internationalized buttons.
- **ActionForm class:** Modified to use ActionServletWrapper rather than expose ActionServlet.
- **ActionServletWrapper class:** Added for use by ActionForm to prevent the Public String properties of ActionServlet from being changed via a query string.
- **Action.MAPPING\_KEY request attribute:** Unconditionally pass the selected mapping as a request attribute ("org.apache.struts.action.mapping.instance") even if no form bean is specified.
- **ActionServlet:** Avoid a NullPointerException in corner cases caused by failed initialization of servlet.
- **ActionForm class:** Now truly serializable, because the two non-serializable instance variables (servlet and multipartRequestHandler) have been made transient. However, if you actually do serialize and deserialize such instances, it is your responsibility to reset these two properties.
- **ActionMessages and ActionErrors:** The initial order a property/key is added in is now retained.

### Upload Package Changes

The following changes and bug fixes have occurred in the file upload package (package `org.apache.struts.upload`):

- **CommonsMultipartRequestHandler:** New implementation of file upload based on the Jakarta Commons FileUpload package. This new implementation is now the default.
- **BufferedMultipartInputStream:** Fixed lost byte problem.
- **ArrayIndexOutOfBoundsException:** Fixed situations where this was known to occur.
- **Multipart requests:** Better reporting for premature closing of input streams while reading multipart requests.
- **New line characters:** Additional fix for file corruption problem with uploads and new line characters.

### Utility Package Changes

The following changes and bug fixes have occurred in the utilities (package `org.apache.struts.util`):

- **RequestUtils:** Added support for forwardPattern, pagePattern, and inputForward properties on ControllerConfig.
- **GenericDataSource:** Deprecated and modified to act as a thin wrapper around `[org.apache.commons.dbpc.BasicDataSource]`. Replaced by direct use of BasicDataSource or other compatible component.
- **RequestUtils class:** Modify to use ActionServletWrapper rather than expose ActionServlet.
- **Added error message for the getActionErrors and getActionMessages method.**
- **getActionErrors and getActionMessages:** Added methods to generate the correct corresponding object based on the object retrieved from request scope based on the key passed in.
- **ActionErrors or ActionMessages:** The logic for creating one of these objects has been moved to a utility method in RequestUtils.
- **JspException message:** Now generated in RequestUtils.
- **ConvertUtils.convertCharacter():** Will now detect empty strings and return the default

value.

### Bean Taglib Package Changes

The following changes and bug fixes have occurred in the *struts-bean* custom tag library (package `org.apache.struts.taglib.bean`):

- `<html:errors>`: When the property tag is specified, errors are no longer printed if the specified property has no errors. Previously errors were always printed ! Future enhancements would include additional attributes to always turn off the header or footer.
- Made the remaining helper methods "protected" rather than "private".

### HTML Taglib Package Changes

The following changes and bug fixes have occurred in the *struts-html* custom tag library (package `org.apache.struts.taglib.html`):

- `FormTag`: Fixed to exclude query string when identifying action mapping name.
- `ImgTag`: Correctly URLEncode the query string parameter value even if there is only a single parameter.
- `MultiboxTag.doAfterBody()`: Corrected to return `SKIP_BODY` instead of `SKIP_PAGE`.

### Logic Taglib Package Changes

The following changes and bug fixes have occurred in the *struts-logic* custom tag library (package `org.apache.struts.taglib.logic`):

- None.

### Documentation Application Changes

The following changes and bug fixes to the Struts Documentation application (and corresponding contents on the Struts web site) have occurred:

- Reorganized Resources into separate pages..
- In the Tag Developers Guide, add more detail regarding file upload requirements.
- In Building View Components, clarify that additional i18n support may be provided by the browser, and is outside the scope of the framework.
- In Building Controller Components, document 'validating' init-param, add defaults for various parameters, clarify that some web.xml settings are not Struts-specific.
- Tag library documentation: Moved under User's Guide.
- Reorganized to separate 1.0 material from nightly build material.

### MailReader Example Application Changes

The following changes and bug fixes to the Struts MailReader Example Application have occurred:

- Add Russian and Japanese translations of the application resources and set the character set for the example JSP pages to "UTF-8" so that it can display either English or Japanese.
- Exchange "name" for "attribute" properties for Edit mappings in Struts configuration file.
- Remove references to saving database data from "tour" document, since this functionality was removed.

## Template Example Application Changes

The following changes and bug fixes to the Struts Template Example Application have occurred:

- None.

## Exercise Taglib Example Application Changes

The following changes and bug fixes to the Struts Exercise Taglib Example Application have occurred:

- Added test case for `<html:link>` using "action" attribute.
- Added test case for `<html:select>` using `<html:options>` based on a collection saved in the page context.

## What's different?

This section provides links to the Struts JavaDoc for any classes that have been added or deprecated since the Struts 1.0 release.

### Previously deprecated classes and packages removed in Struts 1.1

- Removed: `org.apache.struts.utils.BeanUtils`, `org.apache.struts.utils.ConvertUtils`, and `org.apache.struts.utils.PropertyUtils` - replaced by [org.apache.commons.beanutils](#)
- Removed: `org.apache.struts.util.ArrayStack`, `org.apache.struts.util.FastArrayList`, `org.apache.struts.util.FastHashMap`, `org.apache.struts.util.FastTreeMap` - replaced by [org.apache.commons.collections](#)
- Removed: `org.apache.struts.digester.*` - replaced by [org.apache.commons.digester](#)
- Removed: The `struts-config.dtd` - Replaced by [struts-config 1.1.dtd](#).
- Removed: The omnibus "struts" taglib and its associated TLD - replaced by separate bean, logic, and html taglibs.
- Removed: The "form" taglib and its associated TLD - replaced by (renamed as) the html taglib.

### Packages added in Struts 1.1

- [config](#)
- [taglib.nested](#)
- [taglib.nested.bean](#)
- [taglib.nested.html](#)
- [taglib.nested.logic](#)
- [validator](#)

### Classes added in Struts 1.1

action

- [ActionMessage](#)
- [ActionMessages](#)
- [DynaActionForm](#)
- [DynaActionFormClass](#)
- [ExceptionHandler](#)
- [RequestProcessor](#)

actions

- [LookupDispatchAction](#)

taglib.html

- [FrameTag](#)
- [JavascriptValidatorTag](#)
- [MessagesTag](#)
- [OptionsCollectionTag](#)

taglib.logic

- [EmptyTag](#)
- [MessagesNotPresentTag](#)
- [MessagesPresentTag](#)

upload

- [CommonsMultipartRequestHandler](#)

util

- [LabelValueBean](#)

## Classes with members added in Struts 1.1

### [action.Action](#)

- ACTION\_SERVLET\_KEY
- APPLICATION\_KEY
- MESSAGE\_KEY
- PLUG\_INS\_KEY
- REQUEST\_PROCESSOR\_KEY
- execute
- getResources(javax.servlet.http.HttpServletRequest)
- saveMessages

### [action.ActionServlet](#)

- configDigester
- convertHack
- log
- processor
- getInternal

- `destroyApplications`
- `destroyConfigDigester`
- `getApplicationConfig`
- `getRequestProcessor`
- `initApplicationConfig`
- `initApplicationDataSources`
- `initApplicationPlugIns`
- `initApplicationMessageResources`
- `initConfigDigester`
- methods created for backward-compatibility only
  - `defaultControllerConfig`
  - `defaultFormBeansConfig`
  - `defaultForwardsConfig`
  - `defaultMappingsConfig`
  - `defaultMessageResourcesConfig`

#### [taglib.html.BaseHandlerTag](#)

- `indexed`
- `setIndexed`
- `getIndexed`

### **Classes deprecated between Struts 1.0 and Struts 1.1**

#### `action`

- [ActionException](#)
- [ActionFormBeans](#)
- [ActionForwards](#)
- [ActionMappings](#)

### **Classes with members deprecated between Struts 1.0 and Struts 1.1**

#### [action.Action](#)

- `FORM_BEANS_KEY`
- `FORWARDS_KEY`
- `MAPPINGS_KEY`
- `getResources()`
- `perform`

#### [ActionServlet](#)

- `findDataSource`
- `findFormBean`
- `findForward`
- `findMapping`
- `initDataSources`
- methods created for backward-compatibility only
  - `defaultControllerConfig`
  - `defaultFormBeansConfig`
  - `defaultForwardsConfig`
  - `defaultMappingsConfig`
  - `defaultMessageResourcesConfig`



---

*Copyright (c) 2000-2002, Apache Software Foundation*

**Powered by**  
**Struts**



## User Guide

[Table of Contents](#)  
[Preface](#)  
[Introduction](#)  
[Model Components](#)  
[View Components](#)  
[Controller Components](#)  
[Configuration](#)  
[Release Notes](#)  
[Installation](#)

## Developer Guides

[Bean Tags](#)  
[HTML Tags](#)  
[Logic Tags](#)  
[Nested Tags](#)  
[Template Tags](#)  
[Tiles Tags](#)  
[Utilities](#)  
[Validator](#)

## Quick Links

[Welcome](#)  
[News and Status](#)  
[Resources](#)  
[User and Developer Guides \\*](#)  
[FAQs and HowTos](#)

## 6.2 Installation

Contributors:

- Craig R. McClanahan
- Mike Schachter
- Ted Husted
- Martin Cooper
- Chris Assenza
- dIon Gillard
- Eric Wu
- John Rousseau
- John Ueltzhoeffer
- Mark Budai
- Paul Runyan
- Robert Hayden
- Stanley Santiago
- Wong Kok Kai
- Rob Leland
- John Berry

## Prerequisite Software

The Struts binary distribution needs three other software packages installed to operate. You may already have these installed on your system. To build Struts from source you may need to acquire and install several others. The complete list is as follows:

- **Java Development Kit** - You **must** download and install a Java2 (version 1.2 or later) Java Development Kit implementation for your operating system platform. A good starting point for locating Java Development Kit distributions is <http://java.sun.com/j2se>.
- **Servlet Container** - You **must** download and install a servlet container that is compatible with the Servlet API Specification, version 2.2 or later, and the JavaServer Pages (JSP) Specification, version 1.1 or later. One popular choice is to download Apache's [Tomcat](#) (version 3.1 or later required, version 3.3 or later recommended).
- **XML Parser** - Struts **requires** the presence of an XML parser that is compatible with the Java API for XML Parsing (JAXP) specification, 1.1 or later. You can download and install the JAXP [reference implementation](#), which is required for building the Struts source distribution. In Struts-based web applications, you may replace the reference implementation classes with any other JAXP compliant parser, such as [Xerces](#). See detailed instructions related to the parser in the instructions for [building](#) and [installing](#) Struts, below.
- **Ant Build System** - If you are building Struts from the source distribution, you must download and install version 1.5.1 (or later) of the [Ant](#) build system. This package is also strongly recommended for use in developing your own web applications based on Struts.
  - Make sure that the "ant" and "ant.bat" scripts are executable, by adding the \$ANT\_HOME/bin directory to your PATH environment variable.
- **Servlet API Classes** - In order to compile Struts itself, or applications that use Struts, you

will need a `servlet.jar` file containing the Servlet and JSP API classes. Most servlet containers include this JAR file. Otherwise, you can get the Servlet API classes distribution from [here](#).

- **JDBC 2.0 Optional Package Classes** - Struts supports an optional implementation of `javax.sql.DataSource`, so it requires the API classes to be compiled. They can be downloaded from <http://java.sun.com/products/jdbc/download.html>.
- **Jakarta Commons Packages** - Struts utilizes several packages from the [Jakarta Commons Project](#). These are the packages which must be available if you wish to build Struts from source:
  - *Beanutils* (Version 1.5 or later)
  - *Collections* (Version 2.1 or later)
  - *DBCP* (Version 1.0 or later)
  - *Digester* (Version 1.3 or later)
  - *FileUpload* (Recent [Nightly Build](#))
  - *Lang* (Version 1.1 or later)
  - *Logging* (Recent [Nightly Build](#), or Version 1.0.3 or later when it is available)
  - *Pool* (Version 1.0.1 or later)
  - *Validator* (Version 1.0.1 or later)

For your convenience, the requisite JARs are provided as a single download under the `lib` sub-directory with each release and beta distribution. Please note that the minimum requirements may change between releases and betas, and some JARs may need to be updated to use the latest Nightly Build.

- **Jakarta Commons SandBox Packages** - The current version of Struts utilizes several packages from the [Jakarta Commons Sandbox Project](#). Before the final release of Struts 1.1 these packages will be migrated to the Jakarta Commons sections, and officially released. If you are utilizing a binary distribution of Struts, the corresponding JAR files are included in the `lib` directory. However, if you wish to build Struts from source, you will need to download and install the following packages:
  - *Resources* (Recent [Nightly Build](#))
- **Jakarta ORO** - [Apache Jakarta ORO](#) version 2.06 (or later) is required to build Struts from source.
- **Xalan XSLT Processor** - If you are building Struts from the source distribution, you will need a version of Xalan to perform XSLT transformations. If you are using the JAXP/1.1 XML parser, you should use the version of `xalan.jar` shipped with it. Otherwise, download and install version 1.2 of Xalan from [here](#).
- **Cactus Testing** - If you plan on testing the Struts applications in this distribution, you must download and install version 1.3 of the [Cactus](#) test framework. This package is also recommended for use in developing your own unit tests for your web applications based on Struts.

## Install A Struts Binary Distribution

First, download a binary distribution of Struts by following the instructions [here](#). Then, make sure you have downloaded and installed the [prerequisite](#) software packages described above.

Unpack the Struts binary distribution into a convenient directory. (If you [build Struts from the source distribution](#), the result of the build will already be an unpacked binary distribution for you). The distribution consists of the following contents:

- **lib/commons-\*.jar** - These JAR files contain packages from the Jakarta Commons project that are utilized within Struts itself. When you assemble a Struts-based application, you will need to copy these files to the `WEB-INF/lib` directory.
- **lib/jdbc2\_0-stdext.jar** - The JDBC 2.0 Optional Package API classes. You will need to

copy this file to your `WEB-INF/lib` directory if you are utilizing the data sources support provided by Struts.

- **lib/struts.jar** - This JAR file contains all of the Java classes included in Struts. It should be copied into the `WEB-INF/lib` directory of your web application. *WARNING* - If you are going to be hosting multiple Struts based applications on the same servlet container, you will be tempted to place the `struts.jar` file into the shared repository supported by your container. Be advised that this will like cause you to encounter `ClassNotFoundException` problems unless *all* of your application classes are stored in the shared repository.
- **lib/struts-\*.tld** - These are the "tag library descriptor" files that describe the custom tags in the various Struts tag libraries. They should be copied into the `WEB-INF` directory of your web application. (Servlet 2.3 can omit this step if the [standard uri](#) is referenced.)
- **webapps/struts-blank.war** - This is a simple "web application archive" file containing a basic starting point for building your own Struts-based applications.
- **webapps/struts-documentation.war** - This is a "web application archive" file containing all of the Struts documentation found on the [Struts web site](#) (including these pages). You can install this web application on any servlet container compatible with Servlet API 2.2 or later.
- **webapps/struts-example.war** - This is an example web application that uses a large percentage of Struts features. You can install this web application on any servlet container compatible with the Servlet 2.2 (or later) and JSP 1.1 (or later) specifications. If an XML parser is not made available to web applications by your container, you will need to add one to the `WEB-INF/lib` directory of this web application.
- **webapps/struts-exercise-taglib.war** - This web application contains test pages for the various custom tags supported by Struts. It is primarily of use to developers who are enhancing the Struts custom tag libraries, but may also be useful as simple examples of the usage of various Struts tags.
- **webapps/struts-template.war** - This web application both introduces and demonstrates the Struts template tags.
- **webapps/struts-upload.war** - This web application is a quick example of uploading files using the Struts framework.
- **webapps/struts-validator.war** - This web application is an example of using the validator framework, using both the server-side and optional client-side validation.
- **webapps/tiles-documentation.war** - This web application documents how to use tiles, and was developed using tiles.

To use Struts in your own application, you will need to follow these steps:

- Copy the files `lib/commons-*.jar` from the Struts distribution into the `WEB-INF/lib` directory of your web application.
- Copy the file `lib/struts.jar` from the Struts distribution into the `WEB-INF/lib` directory of your web application.
- Copy the all of the files that match `lib/struts-*.tld` from the Struts distribution into the `WEB-INF` directory of your web application.
- Modify the `WEB-INF/web.xml` file for your web application to include a `<servlet>` element to define the controller servlet, and a `<servlet-mapping>` element to establish which request URIs are mapped to this servlet. Use the `WEB-INF/web.xml` file from the Struts example application for a detailed example of the required syntax.
- Modify the `WEB-INF/web.xml` file of your web application to include the following tag library declarations (Servlet 2.3 can omit this step if the [standard uri](#) is referenced):

```
<taglib>
  <taglib-uri>/WEB-INF/struts-bean.tld</taglib-uri>
  <taglib-location>/WEB-INF/struts-bean.tld</taglib-location>
</taglib>
```

```

<taglib>
  <taglib-uri>/WEB-INF/struts-html.tld</taglib-uri>
  <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
</taglib>

<taglib>
  <taglib-uri>/WEB-INF/struts-logic.tld</taglib-uri>
  <taglib-location>/WEB-INF/struts-logic.tld</taglib-location>
</taglib>

<taglib>
  <taglib-uri>/WEB-INF/struts-template.tld</taglib-uri>
  <taglib-location>/WEB-INF/struts-template.tld</taglib-location>
</taglib>

```

- Create a file `WEB-INF/struts-config.xml` that defines the action mappings and other characteristics of your specific application. You can use the `struts-config.xml` file from the Struts example application for a detailed example of the required syntax.
- At the top of each JSP page that will use the Struts custom tags, add line(s) declaring the Struts custom tag libraries used on this particular page, like this:

```

<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>
<%@ taglib uri="/WEB-INF/struts-template.tld" prefix="template" %>

```

- When compiling the Java classes that comprise your application, be sure to include the `struts.jar` and `commons-*.jar` files (copied earlier) on the CLASSPATH that is submitted to the compiler.

## Installing Struts With Your Servlet Container

For most containers, you need only to:

- Copy the WAR files in your Struts `/webapp` directory to your containers `webapps` directory.
- In some cases, you may need to restart your container if it is running.

## Running Struts Applications Under A Security Manager

Many application servers execute web applications under the control of a Java security manager, with restricted permissions on what classes in the web application can do. If you utilize form beans with mapped properties, you may encounter security exceptions unless you add the following permission to the set of permissions granted to your Struts application's codebase:

```
permission java.lang.RuntimePermission "accessDeclaredMembers";
```

Consult the documentation on your application server for more information about how to configure additional security manager permissions.

## Installing Struts on Various Containers

- Bluestone Universal Business Server 7.2 - [Additional steps required.](#)
- Borland Application Server 4.5 -No additional steps required.

- iPlanet Application Server - Service Pack 2 is recommended. Note that the database object in the Struts-Example application is not compatible with this container.
- iPlanet Web Server - [Additional steps required.](#)
- iPortal Application Server - [Additional steps required.](#)
- Jetty - [Additional steps required.](#)
- JRun - [Additional steps required.](#)
- Novell Extensible Application Server 4.0+ - [Additional steps required.](#)
- Orion Application Server - [Additional steps required.](#)
- Resin 1.2+ "standalone" - No additional steps required.
- RexIP - No additional steps required.
- SilverStream 3.7.1 and later - [Additional steps required.](#)
- Tomcat 3.1 and prior - Not recommended. Use Tomcat 3.2.1 or later.
- Tomcat 3.2.1 with Apache - [Additional steps required.](#)
- Tomcat 3.2.1+ "standalone" - No additional steps required.
- Tomcat 4.0 - No additional steps required.
- Weblogic - [Additional steps required.](#)
- WebLogic 6.0+ - No additional steps required.
- WebSphere - [Additional steps required.](#)
- WebSphere - [Steps for the Example Application.](#)

## Building Struts From Source

First, download a source distribution of Struts by following the instructions [here](#). Then, make sure you have downloaded and installed **all** of the [prerequisite](#) software packages described above.

To build Struts, you will need to customize the build process to the details of your development environment as follows:

- The Struts source distribution uses a file named `build.properties` (in the top-level directory of the distribution) to identify the location of external components that Struts depends on.
- There is no `build.properties` file included with the source distribution. However, there is an example file named `build.properties.example` that you can copy to `build.properties` and then customize.
- The properties you must configure in `build.properties` are:
  - **`catalina.home`** - Pathname to the directory of your binary distribution of Tomcat 4.0 (required only if you wish to use the `deploy.catalina` target).
  - **`commons-beanutils.jar`** - Pathname of the BeanUtils package JAR file from the Jakarta Commons project.
  - **`commons-collections.jar`** - Pathname of the Collections package JAR file from the Jakarta Commons project.
  - **`commons-dbcp.jar`** - Pathname of the DBCP package JAR file from the Jakarta Commons project.
  - **`commons-digester.jar`** - Pathname of the Digester package JAR file from the Jakarta Commons project.
  - **`commons-fileupload.jar`** - Pathname of the Fileupload package JAR file from the Jakarta Commons project.
  - **`commons-lang.jar`** - Pathname of the Lang package JAR file from the Jakarta Commons project.
  - **`commons-logging.jar`** - Pathname of the Logging package JAR file from the Jakarta Commons project.
  - **`commons-pool.jar`** - Pathname of the Pool package JAR file from the Jakarta Commons project.

- **commons-resources.jar** - Pathname of the Resources package JAR file from the Jakarta Commons project.
- **commons-validator.jar** - Pathname of the Validator package JAR file from the Jakarta Commons project.
- **servletapi.home** - Pathname to the directory of your binary distribution of the Servlet API classes.
- **tomcat.home** - Pathname to the directory of your binary distribution of Tomcat 3.2 (required only if you wish to use the `deploy.tomcat` target).
- **xerces.home** - Pathname to the directory of your binary distribution of the Xerces parser, version 1.2 or 1.3 (required only if you wish to use the `deploy.catalina` target).
- If you are a Struts developer with write access to the CVS repository, be sure that you do **NOT** check in a copy of the `build.properties` file, since it will be different for each individual developer.

To build a "distribution" version of Struts, first change your current directory to the directory in which you have unpacked the Struts source distribution, and (if necessary) create or customize the `build.properties` file as described above. Then, type:

```
ant dist
```

This command will create a binary distribution of Struts, in a directory named `dist` (relative to where you are compiling from). This directory contains an exact replica of the files included in a binary distribution of Struts, as described in the [preceding section](#).

IMPORTANT NOTE: The `struts.jar`, as well as the JAR files from the Jakarta Commons project, must be in your classpath when compiling Struts. The `build.xml` provided does this automatically. If you use your development machine to test Struts application locally, be sure that the `struts.jar` is **NOT** on your classpath when your container is running.

Next: [FAQs and HowTos](#)





## User Guide

[Table of Contents](#)

[Preface](#)

[Introduction](#)

[Model Components](#)

[View Components](#)

[Controller Components](#)

[Configuration](#)

[Release Notes](#)

[Installation](#)

## Developer Guides

[Bean Tags](#)

[HTML Tags](#)

[Logic Tags](#)

[Nested Tags](#)

[Template Tags](#)

[Tiles Tags](#)

[Utilities](#)

[Validator](#)

## Quick Links

[Welcome](#)

[News and Status](#)

[Resources](#)

[User and Developer Guides \\*](#)

[FAQs and HowTos](#)

## Bean Taglib Guide

Contributors:

- [Craig R. McClanahan](#)

## The Bean Taglib

**Note:** - Some of the features in this taglib are also available in the [JavaServer Pages Standard Tag Library \(JSTL\)](#). The Struts team encourages the use of the standard tags over the Struts specific tags when possible.

The "struts-bean" tag library contains JSP custom tags useful in defining new beans (in any desired scope) from a variety of possible sources, as well as a tag to render a particular bean (or bean property) to the output response.

This tag library contains tags useful in accessing beans and their properties, as well as defining new beans (based on these accesses) that are accessible to the remainder of the page via scripting variables and page scope attributes. Convenient mechanisms to create new beans based on the value of request cookies, headers, and parameters are also provided.

## Bean Taglib Resources

- [Bean Taglib Package Description](#)
- [Bean Taglib API Reference](#)





## User Guide

[Table of Contents](#)

[Preface](#)

[Introduction](#)

[Model Components](#)

[View Components](#)

[Controller Components](#)

[Configuration](#)

[Release Notes](#)

[Installation](#)

## Developer Guides

[Bean Tags](#)

[HTML Tags](#)

[Logic Tags](#)

[Nested Tags](#)

[Template Tags](#)

[Tiles Tags](#)

[Utilities](#)

[Validator](#)

## Quick Links

[Welcome](#)

[News and Status](#)

[Resources](#)

[User and Developer Guides \\*](#)

[FAQs and HowTos](#)

## HTML Taglib Guide

Contributors:

- Craig R. McClanahan
- Ted Husted
- Martin Cooper
- Mike Schachter

## The HTML Taglib

The tags in the Struts HTML library form a bridge between a JSP view and the other components of a Web application. Since a dynamic Web application often depends on gathering data from a user, input forms play an important role in the Struts framework. Consequently, the majority of the HTML tags involve HTML forms.

The HTML taglib contains tags used to create Struts input forms, as well as other tags generally useful in the creation of HTML-based user interfaces.

## HTML Tag Resources

- [HTML Taglib Package Description](#)
- [HTML Taglib API Reference](#)



## User Guide

[Table of Contents](#)

[Preface](#)

[Introduction](#)

[Model Components](#)

[View Components](#)

[Controller Components](#)

[Configuration](#)

[Release Notes](#)

[Installation](#)

## Developer Guides

[Bean Tags](#)

[HTML Tags](#)

[Logic Tags](#)

[Nested Tags](#)

[Template Tags](#)

[Tiles Tags](#)

[Utilities](#)

[Validator](#)

## Quick Links

[Welcome](#)

[News and Status](#)

[Resources](#)

[User and Developer Guides \\*](#)

[FAQs and HowTos](#)

## Logic Taglib Guide

Contributors:

- Craig R. McClanahan
- Mike Schachter

## The Logic Taglib

**Note:** - Many of the features in this taglib are also available in the [JavaServer Pages Standard Tag Library \(JSTL\)](#). The Struts team encourages the use of the standard tags over the Struts specific tags when possible.

The "struts-logic" tag library contains tags that are useful in managing conditional generation of output text, looping over object collections for repetitive generation of output text, and application flow management

## Logic Tag Resources

- [Logic Taglib Package Description](#)
- [Logic Taglib API Reference](#)



## User Guide

[Table of Contents](#)

[Preface](#)

[Introduction](#)

[Model Components](#)

[View Components](#)

[Controller Components](#)

[Configuration](#)

[Release Notes](#)

[Installation](#)

## Developer Guides

[Bean Tags](#)

[HTML Tags](#)

[Logic Tags](#)

[Nested Tags](#)

[Template Tags](#)

[Tiles Tags](#)

[Utilities](#)

[Validator](#)

## Quick Links

[Welcome](#)

[News and Status](#)

[Resources](#)

[User and Developer Guides \\*](#)

[FAQs and HowTos](#)

## Nested Taglib Guide

Contributors:

- Arron Bates

## The Nested Taglib

Nested tags & supporting classes extend the base struts tags to allow them to relate to each other in a nested nature. The fundamental logic of the original tags don't change, except in that all references to beans and bean properties will be managed in a nested context.

## Nested Taglib Resources

- [Nested Taglib Package Description](#)
- [Nested Taglib API Reference](#)



## User Guide

[Table of Contents](#)

[Preface](#)

[Introduction](#)

[Model Components](#)

[View Components](#)

[Controller Components](#)

[Configuration](#)

[Release Notes](#)

[Installation](#)

## Developer Guides

[Bean Tags](#)

[HTML Tags](#)

[Logic Tags](#)

[Nested Tags](#)

[Template Tags](#)

[Tiles Tags](#)

[Utilities](#)

[Validator](#)

## Quick Links

[Welcome](#)

[News and Status](#)

[Resources](#)

[User and Developer Guides \\*](#)

[FAQs and HowTos](#)

## Template Taglib Guide

Contributors:

- David Geary
- Ted Husted

## The Template Taglib

**Note:** As of Struts 1.1 the template library has been deprecated in favor of [Tiles](#).

The "struts-template" tag library contains tags that are useful in creating dynamic JSP templates for pages which share a common format. These templates are best used when it is likely that a layout shared by several pages in your application will change. The functionality provided by these tags is similar to what can be achieved using standard JSP include directive, but are dynamic rather than static.

## Template Taglib Resources

- [Template Taglib Package Description](#)
- [Template Taglib API Reference](#)



## User Guide

[Table of Contents](#)

[Preface](#)

[Introduction](#)

[Model](#)

[Components](#)

[View](#)

[Components](#)

[Controller](#)

[Components](#)

[Configuration](#)

[Release](#)

[Notes](#)

[Installation](#)

## Developer Guides

[Bean Tags](#)

[HTML](#)

[Tags](#)

[Logic](#)

[Tags](#)

[Nested](#)

[Tags](#)

[Template](#)

[Tags](#)

[Tiles Tags](#)

## Tiles Guide

Contributors:

- Cedric Dumoulin

## The Tiles Document Assembly Framework

Tiles builds on the "include" feature provided by the JavaServer Page specification to provide a full-featured, robust framework for assembling presentation pages from component parts. Each part, or tile, can be reused as often as needed throughout your application. This reduces the amount of markup that needs to be maintained and makes it easier to change the look and feel of a website.

## Overview of Tiles Features

- Screen definitions
  - Create a screen by assembling **Tiles**: header, footer, menu, body
  - Definitions can take place :
    - in a centralized xml file
    - directly in jsp page
    - in struts action
  - Definitions provide an inheritance mechanism : a definition can extend another one, and override parameters.
- Templating
  - **Tiles** framework is entirely compatible with *Templates* defined by David Geary and implemented in Struts
  - You can replace *Templates* library by **Tilesone**
- Layouts
  - Define common page layouts and reuse them across your web site
  - Define menu layouts, and use them by passing lists of items and links
  - Define portal layout, use it by passing list of **Tiles** (pages) to show

[Utilities](#)[Validator](#)

## Quick Links

[Welcome](#)[News and](#)[Status](#)[Resources](#)[User and](#)[Developer](#)[Guides \\*](#)[FAQs and](#)[HowTos](#)

- Reuse existing layouts, or define your own
- Dynamic page building
  - Tiles are gathered dynamically during page reload. It is possible to change any attributes: layout, list of Tiles in portal, list of menu items, ...
- Reuse of *Tiles* /Components
  - If well defined, a *Tile* can be reused in different location
  - Dynamic attributes are used to parameterized *Tiles*
  - It is possible to define library of reusable *Tiles* .
  - Build a page by assembling predefined components, give them appropriate parameters
- Internationalization (i18n)
  - It is possible to load different tiles according to Locale
  - A mechanism similar to Java properties files is used for definitions files : you can have one definition file per Locale. The appropriate definition is loaded according to current Locale
- Multi-channels
  - It is possible to load different Tiles according to a key stored in jsp session, or anywhere.
  - For example, key could be user privilege, browser type, ...
  - A mechanism similar to Java properties files is used for definitions files : you can have one definition file per key. The appropriate definition is loaded according to the key.

## Enabling your application for Tiles

The Tiles framework is bundled with Struts. It is not enabled by default. To enable Tiles you need to:

- Setup the struts-tiles taglib in your WEB-INF/web.xml file to include the following tag library declarations:

```
<taglib>
<taglib-uri>/WEB-INF/struts-tiles.tld</taglib-uri>
<taglib-location>/WEB-INF/struts-tiles.tld</taglib-location>
</taglib>
```

- At the top of each JSP page that will use the Tiles custom tags, add line(s) declaring the Tiles custom tag libraries used on this particular page, like this:

```
<%@ taglib uri="/WEB-INF/struts-tiles.tld" prefix="tiles" %>
```

- If you plan to use Tiles definitions defined in a centralized file, you need to create this file, and instruct Struts to load the plugin which will create the factory corresponding to the file. You can have more than one definitions files.
  - Create a file containing your definitions and give it a name (ex: WEB-INF/tiles-defs.xml). You can

use the `tiles-defs.xml` file from the Tiles application for a detailed example of the required syntax.

- Setup Tiles plugin in each `struts-config.xml` file corresponding to a module:

```
<plug-in className="org.apache.struts.tiles.TilesPlugin" >
  <set-property property="definitions-config" value="/WEB-INF/tiles-defs.xml,
/WEB-INF/tiles-tests-defs.xml" />
  <set-property property="definitions-parser-validate" value="true" />
  <set-property property="moduleAware" value="true" />
</plug-in>
```

- **note:** This plugin creates one factory for each Struts modules. The plugin first read factory parameters from `web.xml` and then overload them with parameters from the first `struts-config.xml` file read.
- **note:** Tiles framework now use the commons-logging package to output different information or debug data. Please refer to this package documentation to enable it. The simplest way to enable logging is to create two files under `WEB-INF/classes`:

#### **commons-logging.properties**

```
org.apache.commons.logging.Log=org.apache.commons.logging.impl.SimpleLog
```

#### **simplelog.properties**

```
# Logging detail level,
# Must be one of ("trace", "debug", "info", "warn", "error", or "fatal").
org.apache.commons.logging.simplelog.defaultlog=trace
```

## Tiles API Guide

- A concise [Tiles API Guide](#) is available to help you get started with the Tiles framework.
- [Tiles Taglib Syntax Reference](#)

## Tiles Resources

**Developing applications with Tiles** by Cedric Dumoulin and Ted Husted. Sample chapter from [Struts in Action](#); available as a free download (PDF).

**Using Tiles** Sample beta chapter from [Programming Jakarta Struts](#); available as a free download (PDF).

**Struts and Tiles aid component-based development** by Wellie Chao.

**UI design with Tiles and Struts** by Prakash Malani.

---

*Copyright (c) 2000-2002, Apache Software Foundation*

Powered by  
**Struts**





## User Guide

[Table of Contents](#)

[Preface](#)

[Introduction](#)

[Model Components](#)

[View Components](#)

[Controller Components](#)

[Configuration](#)

[Release Notes](#)

[Installation](#)

## Developer Guides

[Bean Tags](#)

[HTML Tags](#)

[Logic Tags](#)

[Nested Tags](#)

[Template Tags](#)

[Tiles Tags](#)

[Utilities](#)

[Validator](#)

## Quick Links

[Welcome](#)

[News and Status](#)

[Resources](#)

[User and Developer Guides \\*](#)

[FAQs and HowTos](#)

## Utilities Guide

Contributors:

- [Craig R. McClanahan](#)

## The Utilities Package

The Utilities package provides a variety of families of classes, to solve problems that are commonly encountered in building web applications.

## Utilities Resources

- [Utilities Package Description](#)



## User Guide

- [Table of Contents](#)
- [Preface](#)
- [Introduction](#)
- [Model Components](#)
- [View Components](#)
- [Controller Components](#)
- [Configuration](#)
- [Release Notes](#)
- [Installation](#)

## Developer Guides

- [Bean Tags](#)
- [HTML Tags](#)
- [Logic Tags](#)
- [Nested Tags](#)
- [Template Tags](#)
- [Tiles Tags](#)
- [Utilities](#)
- [Validator](#)

## Quick Links

- [Welcome](#)
- [News and Status](#)
- [Resources](#)
- [User and Developer Guides \\*](#)
- [FAQs and HowTos](#)

## Struts Validator Guide

Contributors:

- David Winterfeldt
- James Turner
- Rob Leland

## Struts Validator

The Struts Validator, in some form, has been available since the days of Struts 0.5. It was originally packaged as a developer contribution. Later, the core code was moved to the Jakarta Commons and a Struts specific extension became part of Struts 1.1.

For the convenience of the many developers who have been using the Struts Validator all along, this document first overviews the core functionality and then covers the changes and new functionality added in the Struts 1.1.

Once you have configured the Validator Plug-In, so that it can load your Validator Resources you just have to extend `org.apache.struts.validator.action.ValidatorForm` instead of `org.apache.struts.action.ActionForm`. Then when the `validate` method is called, the action's name attribute from the `struts-config.xml` is used to load the validations for the current form. So the form element's name attribute in the `validator-rules.xml` should match action element's name attribute.

Another alternative is to use the action mapping you are currently on by extending the `ValidatorActionForm` instead of the `ValidatorForm`. The `ValidatorActionForm` uses the action element's `path` attribute from the `struts-config.xml` which should match the form element's name attribute in the `validator-rules.xml`.

Then a separate action can be defined for each page in a multi-page form and the validation rules can be associated with the action and not a page number as in the example of a multi-page form in the validator example.

## Internationalization

Validation rules for forms can be grouped under a `FormSet` element in the `validator-rules.xml` file. The `FormSet` has language, country, and variant attributes that correspond with the `java.util.Locale` class. If they are not used, the `FormSet` will be set to the default locale. A `FormSet` can also have constants associated with it. On the same level as a `FormSet` there can be a global element which can also have constants and have validator actions that perform validations.

The default error message for a pluggable validator can be overridden with the `msg` element. So instead of using the `msg` attribute for the mask validator to generate the error message the `msg` attribute from the field will be used if the name of the field's name attribute matches the validator's name attribute.

```
<field
  property="lastName"
  depends="required,mask">
  <msg
    name="mask"
    key="registrationForm.lastname.maskmsg"/>
  <arg0 key="registrationForm.lastname.displayname"/>
  <var>
    <var-name>mask</var-name>
    <var-value>^[a-zA-Z]*$</var-value>
  </var>
</field>
```

The arguments for error messages can be set with the `arg0-arg3` elements. If the `arg0-arg3` elements' name attribute isn't set, it will become the default arg value for the different error messages constructed. If the name attribute is set, you can specify the argument for a specific pluggable validator and then this will be used for constructing the error message.

```
<field
  property="lastName"
  depends="required,mask">
  <msg
    name="mask"
    key="registrationForm.lastname.maskmsg"/>
  <arg0 key="registrationForm.lastname.displayname"/>
  <var>
    <var-name>mask</var-name>
    <var-value>^[a-zA-Z]*$</var-value>
  </var>
</field>
```

By default the `arg0-arg3` elements will try to look up the key attribute in the message resources. If the resource attribute is set to false, it will pass in the value directly without retrieving the value from the message resources.

```
<field
  property="integer"
  depends="required,integer,range">
  <arg0 key="typeForm.integer.displayname"/>
  <arg1
    name="range"
    key="{var:min}"
    resource="false"/>
  <arg2
    name="range"
    key="{var:max}"
    resource="false"/>
```

```

<var>
  <var-name>min</var-name>
  <var-value>10</var-value>
</var>
<var>
  <var-name>max</var-name>
  <var-value>20</var-value>
</var>
</field>

```

## Constants/Variables

Global constants can be inside the global tags and FormSet/Locale constants can be created in the formset tags. Constants are currently only replaced in the Field's property attribute, the Field's var element value attribute, the Field's msg element key attribute, and Field's arg0-arg3 element's key attribute. A Field's variables can also be substituted in the arg0-arg3 elements (ex: `${var:min}`). The order of replacement is FormSet/Locale constants are replaced first, Global constants second, and for the arg elements variables are replaced last.

```

<global>
  <constant
    name="zip"
    value="^\d{5}(-\d{4})?$" />
</global>

<field
  property="zip"
  depends="required,mask">
<arg0 key="registrationForm.zippostal.displayname"/>
<var>
  <var-name>mask</var-name>
  <var-value>${zip}</var-value>
</var>
</field>

```

The var element under a field can be used to store variables for use by a pluggable validator. These variables are available through the Field's `getVar(String key)` method.

```

<field
  property="integer"
  depends="required,integer,range">
<arg0 key="typeForm.integer.displayname"/>
<arg1
  name="range"
  key="${var:min}"
  resource="false"/>
<arg2
  name="range"
  key="${var:max}"
  resource="false"/>
<var>
  <var-name>min</var-name>
  <var-value>10</var-value>
</var>
<var>
  <var-name>max</var-name>

```

```

        <var-value>20</var-value>
    </var>
</field>

```

## Pluggable Validators

Validation actions are read from the validation.xml file. The default actions are setup in the validation.xml file. The ones currently configured are required, mask, byte, short, int, long, float, double, date (without locale support), and a numeric range.

The 'mask' action depends on required in the default setup. That means that 'required' has to successfully completed before 'mask' will run. The 'required' and 'mask' action are partially built into the framework. Any field that isn't 'required' will skip other actions if the field is null or has a length of zero.

If the [Javascript Tag](#) is used, the client side Javascript generation looks for a value in the validator's javascript attribute and generates an object that the supplied method can use to validate the form. For a more detailed explanation of how the Javascript Validator Tag works, see the [html taglib API reference](#).

The 'mask' action lets you validate a regular expression mask to the field. It uses the Regular Expression Package from the jakarta site. All validation rules can be stored in the validator-rules.xml file. The main class used is `org.apache.regexp.RE`.

Example Validator Configuration from validation.xml.

```

<validator name="required"
    classname="org.apache.struts.validator.util.StrutsValidatorUtil"
    method="validateRequired"
    msg="errors.required"/>

<validator name="mask"
    classname="org.apache.struts.validator.util.StrutsValidatorUtil"
    method="validateMask"
    depends="required"
    msg="errors.invalid"/>

```

### Creating Pluggable Validators

The ValidatorAction method needs to have the following signature.

```

(java.lang.Object,
org.apache.commons.validator.ValidatorAction,
org.apache.commons.validator.Field,
org.apache.struts.action.ActionErrors,
javax.servlet.http.HttpServletRequest,
javax.servlet.ServletContext)

```

- `java.lang.Object` - Bean validation is being performed on.
- `org.apache.commons.validator.ValidatorAction` - The current ValidatorAction being performed.
- `org.apache.commons.validator.Field` - Field object being validated.
- `org.apache.struts.action.ActionErrors` - The errors objects to add an ActionError to if the validation fails.
- `javax.servlet.http.HttpServletRequest` - Current request object.
- `javax.servlet.ServletContext` - The application's ServletContext.

## Multi Page Forms

The field element has an optional page attribute. It can be set to an integer. All validation for the any field page value less than or equal to the current page is performed server side. All validation for the any field page equal to the current page is generated for the client side Javascript. A mutli-part form expects the page attribute to be set.

```
<html:hidden property="page" value="1"/>
```

## Comparing Two Fields

This is an example of how you could compare two fields to see if they have the same value. A good example of this is when you are validating a user changing their password and there is the main password field and a confirmation field.

```
<validator name="twofields"
  classname="com.mysite.StrutsValidator"
  method="validateTwoFields"
  msg="errors.twofields"/>

<field property="password"
  depends="required,twofields">
  <arg0 key="typeForm.password.displayname"/>
  <var>
    <var-name>secondProperty</var-name>
    <var-value>password2</var-value>
  </var>
</field>

public static boolean validateTwoFields(
    Object bean,
    ValidatorAction va,
    Field field,
    ActionErrors errors,
    HttpServletRequest request,
    ServletContext application) {

    String value = ValidatorUtil.getValueAsString(
        bean,
        field.getProperty());
    String sProperty2 = field.getVarValue("secondProperty");
    String value2 = ValidatorUtil.getValueAsString(
        bean,
        sProperty2);

    if (!GenericValidator.isBlankOrNull(value)) {
        try {
            if (!value.equals(value2)) {
                errors.add(field.getKey(),
                    ValidatorUtil.getActionError(
                        application,
                        request,
                        va,
                        field));

                return false;
            }
        }
        catch (Exception e) {
```

```

        errors.add(field.getKey(),
            ValidatorUtil.getActionError(
                application,
                request,
                va,
                field));
        return false;
    }
}

return true;
}

```

## Known Bugs

Since the Struts Validator relies on the Commons Validator, problem reports and enhancement requests may be listed against either product.

- [Struts Validator Bugzilla Reports](#)
- [Commons Validator Bugzilla Reports](#)

## Changes and deprecations

### New tag attributes.

The [<html:javascript>](#) tag has new attributes defined.

### Validating against the DTD in the commons-validator.jar.

The validator xml files now **validates against the DTD stored in the commons-validator.jar** ! Struts no longer maintains a separate dtd for validator-rules.xml and validator.xml. Additionally, commons-validator now maintains a unified validator.dtd. Change all validator.xml DTD references to:

```

<!DOCTYPE form-validation PUBLIC
"-//Apache Software Foundation//DTD Commons Validator Rules Configuration 1.0//EN"
"http://jakarta.apache.org/commons/dtds/validator_1_0.dtd">

```

### Blank fields.

The default validator-rules.xml now ignores blank fields for all the basic validation types. If you require a field to be present then to your applications validator.xml field definition add "required" to the depends attribute.

### New range methods.

intRange & floatRange methods in both JavaScript and Java

### Conditionally required fields.

The most fundamental change is the ability to conditionally require validator fields based on the value of other fields. It allows you to define logic like "only validate this field if field X is non-null and field Y equals 'male'".

The syntax looks like this:

If you have this in your struts-config.xml

```
<form-bean
  name="dependentlistForm"
  type="org.apache.struts.webapp.validator.forms.ValidatorForm">
  <form-property
    name="dependents"
    type="org.apache.struts.webapp.validator.Dependent[]"
    initial="{ '' , '' , '' , '' , '' , '' , '' , '' , '' , '' , '' , '' }"/>
  <form-property
    name="insureDependents"
    type="java.lang.Boolean"
    initial="false"/>
</form-bean>
```

Where dependent is a bean that has properties lastName, firstName, dob, coverageType

You can define a validation:

```
<form name="dependentlistForm">

<field
  property="firstName" indexedListProperty="dependents"
  depends="requiredif">
<arg0 key="dependentlistForm.firstName.label"/>
<var>
  <var-name>field[0]</var-name>
  <var-value>lastName</var-value>
</var>
<var>
  <var-name>field-indexed[0]</var-name>
  <var-value>true</var-value>
</var>
<var>
  <var-name>field-test[0]</var-name>
  <var-value>NOTNULL</var-value>
</var>
</field>

<field
  property="dob"
  indexedListProperty="dependents"
  depends="requiredif,date">
<arg0 key="dependentlistForm.dob.label"/>
<var>
  <var-name>field[0]</var-name>
  <var-value>lastName</var-value>
</var>
<var>
  <var-name>field-indexed[0]</var-name>
  <var-value>true</var-value>
</var>
<var>
  <var-name>field-test[0]</var-name>
  <var-value>NOTNULL</var-value>
</var>
```



```

</field>

<field
  property="coverageType"
  indexedListProperty="dependents"
  depends="requiredif">
<arg0 key="dependentlistForm.coverageType.label"/>
<var>
  <var-name>field[0]</var-name>
  <var-value>lastName</var-value>
</var>
<var>
  <var-name>field-indexed[0]</var-name>
  <var-value>>true</var-value>
</var>
<var>
  <var-name>field-test[0]</var-name>
  <var-value>NOTNULL</var-value>
</var>
<var>
  <var-name>field[1]</var-name>
  <var-value>insureDependents</var-value>
</var>
<var>
  <var-name>field-test[1]</var-name>
  <var-value>EQUAL</var-value>
</var>
<var>
  <var-name>field-value[1]</var-name>
  <var-value>>true</var-value>
</var>
<var>
  <var-name>field-join</var-name>
  <var-value>AND</var-value>
</var>
</field>

</form>

```

Which is read as follows: The firstName field is only required if the lastName field is non-null. Since field-indexed is true, it means that lastName must be a property of the same indexed field as firstName. Same thing for dob, except that we validate for date if not blank.

The coverageType is only required if the lastName for the same indexed bean is not null, and also if the non-indexed field insureDependents is true.

You can have an arbitrary number of fields by using the [n] syntax, the only restriction is that they must all be AND or OR, you can't mix.

### Deprecations.

- Deprecation of range methods in both JavaScript and Java.
- Deprecation of StrutsValidator & StrutsValidatorUtil.

A concise [Struts Validator API Guide](#) is available to help you get started.

## Validator Resources

**[DynaForms and the Validator](#)** by James Turner and Kevin Bedell. Sample chapter from [Struts Kickstart](#); available as a free download (PDF).

**[Validating user input](#)** by David Winterfeldt and Ted Husted. Sample chapter from [Struts in Action](#); available as a free download (PDF).

---

*Copyright (c) 2000-2002, Apache Software Foundation*

Powered by  
**Struts**



## User Guide

[Table of Contents](#)

[Preface](#)

[Introduction](#)

[Model](#)

[Components](#)

[View](#)

[Components](#)

[Controller](#)

[Components](#)

[Configuration](#)

[Release Notes](#)

[Installation](#)

## Developer Guides

[Bean Tags](#)

[HTML Tags](#)

[Logic Tags](#)

[Nested Tags](#)

[Template Tags](#)

[Tiles Tags](#)

[Utilities](#)

[Validator](#)

## Quick Links

[Welcome](#)

[News and Status](#)

[Resources](#)

[User and](#)

[Developer](#)

[Guides \\*](#)

[FAQs and](#)

[HowTos](#)

## 5.2 Installation

Contributors:

- Stanley Santiago

## Installing Struts with your servlet container

### iPlanet Application Server 6.0

Service Pack 2 is recommended.

NOTE: At present, the Struts example application still uses a non-Serializable servlet context attribute, and will not run in an environment that requires them, like iPlanet Application Server.

### iPlanet Web Server 4.2

Here are the issues I ran into while moving my struts based application from Tomcat (supports WebApps and WAR) to iWS 4.1 (does **NOT** support Webapps and WAR).

Webapps and WAR will be supported in iWS 5.0, as mentioned in iWS5.0 roadmap.

#### Classpath issues.

This is pretty straightforward. Since there is no notion of `WEB-INF/lib` and `WEB-INF/classes` the classpath has to be explicitly set in `$SERVER_ROOT/config/jvm12.conf`.

#### Context relative paths

All URLs should be visible from the document root. In my case I just created a symbolic link from `$DOCR00T/myapp` to `webapps/myapp`.

#### Extension mapping

The config file `$SERVER_ROOT/config/rules.properties` has a similar mechanism as in `web.xml`.

I have this in my `rules.properties` which forwards all urls ending with "do" to the servlet whose logical name is action.

```
####
    @.*[.]do$action
####
```

Back to [Installation](#)

---

*Copyright (c) 2000-2002, Apache Software Foundation*

Powered by  
**Struts**



## User Guide

[Table of Contents](#)

[Preface](#)

[Introduction](#)

[Model](#)

[Components](#)

[View](#)

[Components](#)

[Controller](#)

[Components](#)

[Configuration](#)

[Release Notes](#)

[Installation](#)

## Developer Guides

[Bean Tags](#)

[HTML Tags](#)

[Logic Tags](#)

[Nested Tags](#)

[Template Tags](#)

[Tiles Tags](#)

[Utilities](#)

[Validator](#)

## Quick Links

[Welcome](#)

[News and Status](#)

[Resources](#)

[User and  
Developer  
Guides \\*](#)

[FAQs and  
HowTos](#)

## 5.2 Installation

Contributors:

- John Ueltzhoeffer

## Installing Struts with your servlet container

### iPortal Application Server 1.3

**Tested with: Windows 2000**

#### Important Note:

At the moment, iPAS 1.3 is not fully compliant with the JSP 1.1/1.2 specification.

Specifically, the automatic type conversions for custom tag parameters specified in "Issue 7" of the JSP 1.1 Errata and in the JSP 1.2 Proposed Final Draft have not yet been implemented.

As it stands, JSP pages that make use of Struts taglibs whose parameters require conversion (such as booleans) will not compile under JRun. This includes the Struts Example Application. Attempting to run the example application will result in an exception similar to the following being thrown:

/struts-example/index.jsp:

```
Compilation failed [IT_Builder:1000]
  at com.ionaj2ee.builder.JavaBuilder.build(JavaBuilder.java:84)
  at com.ionaj2ee.builder.JspBuilder.build(JspBuilder.java:51)
  at com.ionaj2ee.builder.WarBuilder.build(WarBuilder.java:111)
  at com.ionaj2ee.builder.EarBuilder.build(EarBuilder.java:99)
  at com.ionaj2ee.builder.EarBuilder.main(EarBuilder.java:223)
  at iportal.build.main(build.java:14)
ocale(boolean) in org.apache.struts.taglib.html.HtmlTag
  cannot be applied to (java.lang.String)
      _x0.setLocale("true");
              ^
1 error
```

(For more details see refer to:

<http://www.mail-archive.com/struts-user@jakarta.apache.org/msg01860.html> )

The following instructions describe how to install the Struts Example Application under iPAS 1.3. A subsequent section describes how the Struts Example Application can be patched to work with Struts.

The following instructions assume the following:

- iPortal Application Server 1.3 has been installed.
- Both the Strut and XML Parser libraries are in your classpath.

### Installing the struts example application

- Start iPAS Services by clicking on the [Start iPAS Services] menu item.
- Start the iPortal Application Server by clicking on the [iPortal Application Server] menu item.
- Start a command shell. Change to the `$INSTALLDIR\IONA` and run the `setenvs.bat` file.
- Create a directory called `jars`.

Now run the EARSCO tool. Type `java iportal.earsco` and at the prompts do:

- Next
- Type in the application name of `struts-example` then click next.
- In step three click the check box and enter the name of the WAR `struts-example`. Then click next.
- Click on Finish.

Now you must copy the contents of the struts-example war into the EARSCO directory structure as follows:

Under `$INSTALLDIR\IONA\jars\struts-examples\src\struts-example.war` you copy contents into the following directories: `etc`, `lib`, `src` and `web`.

- Copy all files in the root directory  
`$INSTALLDIR\jakarta-tomcat-3.2.1\webapps\struts-example\WEB-INF`  
into the earsco directory  
`$INSTALLDIR\IONA\jars\struts-examples\src\struts-example.war\etc`  
Do not copy in the classes or lib directories.
- Copy the directory  
`$INSTALLDIR\jakarta-tomcat-3.2.1\webapps\struts-example\WEB-INF\lib`  
into the earsco directory  
`$INSTALLDIR\IONA\jars\struts-examples\src\struts-example.war\lib`
- Copy the directory  
`$INSTALLDIR\jakarta-tomcat-3.2.1\webapps\struts-example\WEB-INF\classes`  
into the earsco directory  
`$INSTALLDIR\IONA\jars\struts-examples\src\struts-example.war\src`
- Copy the directory  
`$INSTALLDIR\jakarta-tomcat-3.2.1\webapps\struts-example`  
into the earsco directory  
`$INSTALLDIR\IONA\jars\struts-examples\src\struts-example.war\web`
- Next modify the `application.xml` in the  
`$INSTALLDIR\IONA\jars\struts-examples\etc` directory to this:

`<application>`

```
<!-- Add display name -->
    <display-name>Struts Example</display-name>
    . . . . .
```

- Last update the cc.xml in the \$INSTALLDIR\IONA\jars\struts-examples directory as follows:

```
<configuration>
  <web-app>
    <context-root>struts-example</context-root>
  </web-app>
</configuration>
```

Now you are ready to compile and deploy the struts-example.

To compile the source from the \$INSTALLDIR\IONA\jars\struts-examples type

```
java iportal.build
```

Next, type

```
java iportal.deploy
```

The first time you deploy you will be prompted by a Deploy wizard and asked to supply both locations of the struts-example.ear file and of the cc.xml file. Once both elements have been satisfied continue until the finish button and click it. The EAR file should deploy successfully.

Test the sample application by using the following URL in the browser:

```
http://hostname:9000/struts-example/index.jsp
```

The struts-documentation.war can be installed using the same procedure.

### Patching the struts example application

As mentioned at the beginning of these notes, the Struts Example Application will not run under iPAS 1.3 without modification. The following changes will need to be made:

- index.jsp, logon.jsp: Change <html:html locale="true"> to <html:html locale=<%= true %>>
- registration.jsp, subscription.jsp: Change all instances of filter="true" to filter=<%= true %>

---

Back to [Installation](#)



## User Guide

[Table of](#)[Contents](#)[Preface](#)[Introduction](#)[Model](#)[Components](#)[View](#)[Components](#)[Controller](#)[Components](#)[Configuration](#)[Release](#)[Notes](#)[Installation](#)

## Developer Guides

[Bean Tags](#)[HTML](#)[Tags](#)[Logic](#)[Tags](#)[Nested](#)[Tags](#)[Template](#)[Tags](#)[Tiles Tags](#)[Utilities](#)[Validator](#)

## Quick Links

[Welcome](#)[News and](#)[Status](#)[Resources](#)[User and](#)[Developer](#)[Guides \\*](#)[FAQs and](#)[HowTos](#)

## 5.2 Installation

Contributors:

- Paul Runyan

## Installing Struts with your servlet container

### Jetty Java HTTP Servlet Server

Jetty is a small, pure-Java, open source HTTP server that supports the 2.2 Servlet spec and JSP 1.1. Jetty can be downloaded from <http://www.mortbay.com/jetty>.

Struts WAR files run nearly straight out of the box when placed underneath Jetty's webapps directory. The one additional step needed is to add an entry for each WAR file to the Jetty server configuration file in order to map the appropriate request paths to the added Struts web applications (using "<Call name='addWebApplication'>...").

So for example, if you have copied the WAR files that come with the Struts binary distribution into a subdirectory of the "%JETTY\_HOME%/webapps" called "%JETTY\_HOME%/webapps/struts" so that you have:

- %JETTY\_HOME%/webapps/struts/struts-documentation.war
- %JETTY\_HOME%/webapps/struts/struts-example.war
- %JETTY\_HOME%/webapps/struts/struts-exercise-taglib.war
- %JETTY\_HOME%/webapps/struts/struts-template.war
- %JETTY\_HOME%/webapps/struts/struts-upload.war
- %JETTY\_HOME%/webapps/struts/struts-blank.war

And you want to run Jetty using the demo.xml configuration file that comes with Jetty, just add the following block to demo.xml, anywhere after the Listeners are declared.

```
<!-- Jetty config for Struts BEGIN -->

<Call name="addWebApplication">
  <Arg>/struts/struts-documentation/*</Arg>
  <Arg><SystemProperty name="jetty.home"
    default="."/>/webapps/struts/struts-documentation.war</Arg>
  <Arg><SystemProperty name="jetty.home"
    default="."/>/etc/webdefault.xml</Arg>
  <Arg type="boolean">false</Arg> <!-- if true,
    expand war in temp dir -->
</Call>

<Call name="addWebApplication">
  <Arg>/struts/struts-example/*</Arg>
  <Arg><SystemProperty name="jetty.home"
```



```

        default="." />/webapps/struts/struts-example.war</Arg>
<Arg><SystemProperty name="jetty.home"
    default="." />/etc/webdefault.xml</Arg>
<Arg type="boolean">true</Arg> <!-- if true,
    expand war in temp dir -->
</Call>

<Call name="addWebApplication">
<Arg>/struts/struts-exercise-taglib/*</Arg>
<Arg><SystemProperty name="jetty.home"
    default="." />/webapps/struts/struts-exercise-taglib.war</Arg>
<Arg><SystemProperty name="jetty.home"
    default="." />/etc/webdefault.xml</Arg>
<Arg type="boolean">false</Arg> <!-- if true,
    expand war in temp dir -->
</Call>

<Call name="addWebApplication">
<Arg>/struts/struts-template/*</Arg>
<Arg><SystemProperty name="jetty.home"
    default="." />/webapps/struts/struts-template.war</Arg>
<Arg><SystemProperty name="jetty.home"
    default="." />/etc/webdefault.xml</Arg>
<Arg type="boolean">true</Arg> <!-- if true,
    expand war in temp dir -->
</Call>

<Call name="addWebApplication">
<Arg>/struts/struts-upload/*</Arg>
<Arg><SystemProperty name="jetty.home"
    default="." />/webapps/struts/struts-upload.war</Arg>
<Arg><SystemProperty name="jetty.home"
    default="." />/etc/webdefault.xml</Arg>
<Arg type="boolean">true</Arg> <!-- if true,
    expand war in temp dir -->
</Call>

<Call name="addWebApplication">
<Arg>/struts/struts-blank/*</Arg>
<Arg><SystemProperty name="jetty.home"
    default="." />/webapps/struts/struts-blank.war</Arg>
<Arg><SystemProperty name="jetty.home"
    default="." />/etc/webdefault.xml</Arg>
<Arg type="boolean">true</Arg> <!-- if true,
    expand war in temp dir -->
</Call>

<!-- Jetty config for Struts END -->

```

---

Back to [Installation](#)

---





## User Guide

- [Table of Contents](#)
- [Preface](#)
- [Introduction](#)
- [Model Components](#)
- [View Components](#)
- [Controller Components](#)
- [Configuration](#)
- [Release Notes](#)
- [Installation](#)

## Developer Guides

- [Bean Tags](#)
- [HTML Tags](#)
- [Logic Tags](#)
- [Nested Tags](#)
- [Template Tags](#)
- [Tiles Tags](#)
- [Utilities](#)
- [Validator](#)

## Quick Links

- [Welcome](#)
- [News and Status](#)
- [Resources](#)
- [User and Developer Guides \\*](#)
- [FAQs and HowTos](#)

## 5.2 Installation

Contributors:

- Eric Wu
- Mark Budai

## Installing Struts with your servlet container

### JRUN 3.0 SP2A, VERSION 3.02A.11614

**Tested with: Microsoft IIS 5.0, Windows 2000**

#### Important Note:

At the moment, JRun is not fully compliant with the JSP 1.1/1.2 specification.

Specifically, the automatic type conversions for custom tag parameters specified in "Issue 7" of the JSP 1.1 Errata and in the JSP 1.2 Proposed Final Draft have not yet been implemented.

As it stands, JSP pages that make use of Struts taglibs whose parameters require conversion (such as booleans) will not compile under JRun. This includes the Struts Example Application. Attempting to run the example application will result in an exception similar to the following being thrown:

```
/struts-example/index.jsp:
javax.servlet.ServletException: Compilation error occurred:
allaire.jrun.scripting.DefaultCFE:
Errors reported by compiler:
c:/JRun/servers/default/Struts
  Example/WEB-INF/jsp/jrun__index2ejspa.java:41:1:41:27:
Error: No match was found for method "setLocale(java.lang.String)".
```

(For more details see refer to:

<http://www.mail-archive.com/struts-user@jakarta.apache.org/msg01860.html>)

The following instructions describe how to install the Struts Example Application under JRun. A subsequent section describes how the Struts Example Application can be patched to work with Struts

The following instructions assume the following:

- JRun has been installed and integrated with the web server of choice.

- \$APP\_SERVER\_NAME is the name of the application server used to host the application. (When JRun is first installed, it creates an application server called JRun Default Server).
- \$APP\_SERVER\_DIR is the directory used to hold applications hosted by \$APP\_SERVER\_NAME. For the JRun Default Server, the directory is \$JRUN\_HOME/servers/default where \$JRUN\_HOME is the directory where JRun is installed.

## Installing the struts example application

- Login to the JRun Management Console.
- On the left pane, select \$APP\_SERVER\_NAME. A page showing the current server status will be shown on the right pane.
- On the right pane, click on the WAR Deployment link. A page containing a list of the currently deployed web applications will be shown.
- On the right pane, click on Deploy an Application. Complete the Web Application Information form as follows:
  - Servlet War File or Directory: Enter the full path where struts-example.war is found or click on Browse to select the path.
  - JRun Server Name: \$APP\_SERVER\_NAME
  - Application Name: Struts Example
  - Application Hosts: All Hosts
  - Application URL: /struts-example
  - Application Deploy Directory: will default to \$APP\_SERVER\_NAME/Struts Example (or the name as specified for Application Name).
- Once the form is complete, click on the Deploy button.
- If deployment is successful, restart the application server by clicking on \$APP\_SERVER\_NAME on the left pane. A page showing the current server status will be shown on the right pane. Click the Restart Server button to restart the application server.
- Test the sample application by using the following URL in the browser:  
`http://hostname/struts-example/index.jsp`  
The struts-documentation.war can be installed using the same procedure.

## Installing unpacked web applications

The above steps should be followed for applications deployed as \*.war files.

For unpacked web applications, configuration involves the following steps:

- From the JRun Management Console, select \$APP\_SERVER\_NAME (on the left pane) and click on WAR Deployment (on the right pane).
- On the right pane, click on Create an Application and complete the Web Application Information form as follows:
- JRun Server Name: \$APP\_SERVER\_NAME
  - Application Name: myApplication
  - Application Hosts: All Hosts
  - Application URL: /myApplication
  - Application Deploy Directory: will default to \$APP\_SERVER\_NAME/myApplication
- Click on Create to submit the form.
- Once the web application is created, install and configure the struts components (struts.jar, struts\*.tld, etc) for the web application under \$APP\_SERVER\_NAME/myApplication/WEB-INF
- Install the remaining components of the application: .class files, JSP pages, .properties files etc as required.
- To configure the extension mapping of the request URI (ie \*.do) to the action servlet, expand \$APP\_SERVER\_NAME on the left pane, expand the Web Applications branch and click on

myApplication. The right pane will display the configuration options for myApplication. Click on Servlet URL Mappings. A list of existing mappings will be shown. Click the Edit button and create the following entry:

- Virtual Path/Extension: \*.do
- Servlet Invoked: action
- Click on the Update button to save the changes.
- Restart the application server.
- The application should now be accessible from the browser.

The JRun application server will need to be restarted each time one of the following changes are made to the web application:

- .class or .jar files are modified
- .properties files are modified
- .xml files are modified

## Patching the struts example application

As mentioned at the beginning of these notes, the Struts Example Application will not run under JRun without modification. The following changes will need to be made:

- index.jsp, logon.jsp: Change `<html:html locale="true">` to `<html:html locale=<%= true %>>`
- logon.jsp: Change `<html:html redisplay="true">` to `<html:html redisplay=<%= true %>>`
- registration.jsp, subscription.jsp: Change all instances of `filter="true"` to `filter=<%= true %>`

- 
- Author: Eric Wu, Mark Budai
- 

Back to [Installation](#)



## User Guide

[Table of Contents](#)[Preface](#)[Introduction](#)[Model](#)[Components](#)[View](#)[Components](#)[Controller](#)[Components](#)[Configuration](#)[Release Notes](#)[Installation](#)

## Developer Guides

[Bean Tags](#)[HTML Tags](#)[Logic Tags](#)[Nested Tags](#)[Template Tags](#)[Tiles Tags](#)[Utilities](#)[Validator](#)

## Quick Links

[Welcome](#)[News and Status](#)[Resources](#)[User and  
Developer  
Guides \\*](#)[FAQs and](#)[HowTos](#)

## 5.2 Installation

Contributors:

### Installing Struts with your servlet container

#### Orion Application Server

In the steps below, \$ORION\_HOME refers to the directory in which you have installed Orion, and \$STRUTS\_HOME is the directory in which you unpacked the Struts binary distribution.

- Modify the file \$ORION\_HOME/config/application.xml to define the two new applications, by adding the following declarations, immediately following the web-module directive for the default web application:

```
<web-module id="strutsDoc"
path="$STRUTS_HOME/webapps/struts-documentation.war"/>
<web-module id="strutsExample"
path="$STRUTS_HOME/webapps/struts-example.war"/>
```

- Modify the file \$ORION\_HOME/config/default-web-site.xml (or the configuration file for any other Orion web site) to include the following declarations, after the declaration for the <default-web-app> if any:

```
<web-app application="default" name="strutsDoc"
root="/struts-documentation"/>
<web-app application="default" name="strutsExample"
root="/struts-example"/>
```

- After you start Orion, you should now be able to access these applications (assuming you haven't changed the port number from the default of 80) at:

```
http://localhost/struts-documentation
http://localhost/struts-example
```

- Versions of Orion up to at least 1.0.3 have a bug related to ServletContext.getResource() calls that prevent the Struts example application from working out of the box. This manifests itself as a JSP error when you try to access the example application, with the following message:  

```
javax.servlet.jsp.JspException: Missing resources
attributeorg.apache.struts.action.MESSAGE
```

followed by an error traceback. There will also be an initialization error message in the \$ORION\_HOME/log/global-application.log log file. To work around this problem, you can take the following steps:
  - Go to the \$STRUTS\_HOME/webapps directory, where you will note that Orion has automatically expanded each web application into an unpacked directory

structure.

- Go to the `$STRUTS_HOME/webapps/struts-example/WEB-INF` directory, and copy the file `struts-config.xml` one directory up (that is, into `$STRUTS_HOME/webapps/struts-example`).
- Modify the `$STRUTS_HOME/webapps/struts-example/WEB-INF/web.xml` file, changing the value of the "config" initialization parameter (for the action servlet) from `/WEB-INF/struts-config.xml` to `/action.xml`.
- Restart Orion, and you should be able to access the example application.
- Note that this workaround has a negative security-related side effect: your `struts-config.xml` file can now be retrieved by remote clients at the following URL:  
`http://localhost/struts-example/struts-config.xml`  
Therefore, you should be sure you do not store sensitive information (such as database passwords) in this file.

---

Back to [Installation](#)

---

*Copyright (c) 2000-2002, Apache Software Foundation*

**Powered by**  
**Struts**



## User Guide

[Table of](#)[Contents](#)[Preface](#)[Introduction](#)[Model](#)[Components](#)[View](#)[Components](#)[Controller](#)[Components](#)[Configuration](#)[Release](#)[Notes](#)[Installation](#)

## Developer Guides

[Bean Tags](#)[HTML](#)[Tags](#)[Logic](#)[Tags](#)[Nested](#)[Tags](#)[Template](#)[Tags](#)[Tiles Tags](#)[Utilities](#)[Validator](#)

## Quick Links

[Welcome](#)[News and](#)[Status](#)[Resources](#)[User and](#)[Developer](#)[Guides \\*](#)[FAQs and](#)[HowTos](#)

## 5.2 Installation

Contributors:

- John Rousseau

## Installing Struts with your servlet container

### SilverStream Application Server 3.7.1 and later

- Start the SilverStream application server.
- Create an XML deployment plan for the "struts-example.war" application. Call the file "struts-example-depl-plan.xml". You can use the following contents for the file

----- cut here -----

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE warJarOptions PUBLIC
    "-//SilverStream Software, Inc.//DTD J2EE WAR Deployment Plan//EN"
    "deploy_war.dtd">
<warJarOptions>
<warJar>
<warJarName>struts-example.war</warJarName>
<isEnabled>true</isEnabled>
<urls><el>struts-example</el></urls>
</warJar>
</warJarOptions>
```

----- cut here -----

Create an XML deployment plan for the "struts-documentation.war" application. Call the file "struts-documentation-depl-plan.xml". You can use the following contents for the file:

----- cut here -----

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE warJarOptions PUBLIC
    "-//SilverStream Software, Inc.//DTD J2EE WAR Deployment Plan//EN"
    "deploy_war.dtd">
<warJarOptions>
```



```
<warJar>
<warJarName>struts-documentation.war</warJarName>
<isEnabled>true</isEnabled>
<urls><el>struts-documentation</el></urls>
</warJar>
</warJarOptions>
```

----- cut here -----

---

Run the following "SilverCmd DeployWAR" commands to deploy the applications. You can change 'localhost' to whatever server you are deploying to. You can change 'Silvermaster' to whatever database you are deploying to.

- SilverCmd DeployWar localhost Silvermaster struts-example.war -f struts-example-depl-plan.xml
- SilverCmd DeployWar localhost Silvermaster struts-documentation.war -f struts-documentation-depl-plan.xml

- 
- Author: John Rousseau
- 

Back to [Installation](#)



## User Guide

[Table of Contents](#)  
[Preface](#)  
[Introduction](#)  
[Model Components](#)  
[View Components](#)  
[Controller Components](#)  
[Configuration](#)  
[Release Notes](#)  
[Installation](#)

## Developer Guides

[Bean Tags](#)  
[HTML Tags](#)  
[Logic Tags](#)  
[Nested Tags](#)  
[Template Tags](#)  
[Tiles Tags](#)  
[Utilities](#)  
[Validator](#)

## Quick Links

[Welcome](#)  
[News and Status](#)  
[Resources](#)  
[User and Developer Guides \\*](#)  
[FAQs and HowTos](#)

## 5.2 Installation

Contributors:

- Craig R. McClanahan

## Installing Struts with your servlet container

### Tomcat 3.2.1 With Apache

Note that the instructions for Tomcat 4 will be different than those for Tomcat 3, but the Tomcat 4.0 web connector is still under development. Versions of Tomcat prior to 3.2.1 are not recommend for use with Struts.

- These instructions assume you have successfully integrated Tomcat with Apache according to the Tomcat documentation.
- Copy "struts-documentation.war" and "struts-example.war" to your \$TOMCAT\_HOME/webapps directory
- Restart Tomcat if it is already running
- Tomcat will generate a file "\$TOMCAT\_HOME/conf/tomcat-apache.conf" that will be used by Apache. This file is regenerated every time you start Tomcat, so copy this file to a safe place (such as your Apache configuration directory; on Unix systems this is usually /usr/local/apache/conf.
- If you are running Tomcat 3.1, Tomcat will not have generated the entries for your new applications. Add the following lines to the tomcat-apache.conf file that you have saved, replacing \$TOMCAT\_HOME with the path to your Tomcat home directory:

```
Alias /struts-documentation "$TOMCAT_HOME/webapps/struts-documentation"
<Directory "$TOMCAT_HOME/webapps/struts-documentation">
    Options Indexes FollowSymLinks
</Directory>
    ApJServMount /struts-documentation/servlet /struts-documentation
<Location "/struts-documentation/WEB-INF/">
    AllowOverride None
    deny from all
</Location>
Alias /struts-example "$TOMCAT_HOME/webapps/struts-example"
<Directory "$TOMCAT_HOME/webapps/struts-example">
    Options Indexes FollowSymLinks
</Directory>
    ApJServMount /struts-example/servlet /struts-example
<Location "/struts-example/WEB-INF/">
    AllowOverride None
    deny from all
</Location>
```

- The generated file above does not know anything about extension mappings defined in a web.xml

file, so the "\*.do" URIs that go to the controller servlet will not be recognized. To fix this, add the following line to the saved version of "tomcat-apache.conf", after the corresponding line for the .jsp extension:

```
AddHandler jserv-servlet .do
```

- Ensure that the saved version of "tomcat-apache.conf" is referenced in your Apache "httpd.conf" configuration file. A typical use would have the following line at the bottom of "httpd.conf":

```
Include /usr/local/apache/conf/tomcat-apache.conf
```

- In order to recognize "index.jsp" as a default page for web applications, search in your "httpd.conf" for a "DirectoryIndex" directive. If you have one, add "index.jsp" to the end of the list, so that it might look like this:

```
DirectoryIndex index.html index.jsp
```

If you do not have such an entry, add one like this:

```
DirectoryIndex index.jsp
```

- Restart Apache to make it aware of the new applications. You should now be able to access the applications from a browser like this:

```
http://localhost/struts-documentation
```

```
http://localhost/struts-example
```

- 
- Author: Craig R. McClanahan
- 

Back to [Installation](#)



## User Guide

[Table of Contents](#)[Preface](#)[Introduction](#)[Model](#)[Components](#)[View](#)[Components](#)[Controller](#)[Components](#)[Configuration](#)[Release Notes](#)[Installation](#)

## Developer Guides

[Bean Tags](#)[HTML Tags](#)[Logic Tags](#)[Nested Tags](#)[Template Tags](#)[Tiles Tags](#)[Utilities](#)[Validator](#)

## Quick Links

[Welcome](#)[News and Status](#)[Resources](#)[User and  
Developer  
Guides \\*](#)[FAQs and  
HowTos](#)

## 5.2 Installation

Contributors:

- Mike Schachter

## Installing Struts with your servlet container

### Bluestone Universal Business Server 7.2

- You need UBS version 7.2 to run war file applications. The UBS 7.2.2 evaluation is located [here](#). If you're using version 7.2.1, you need to download the WAR file patch, located in the product enhancement section of Bluestone's website [here](#)
- After installation of the correct version and/or patch of UBS 7.2, you need to modify your apserver.txt file to point to the correct directory for your war file applications. Look for the section that says something similar to the following:

```
[SaServletEngine.class]
session_affinity=1
type=1
program=/SaServletEngine.class
file_path=f:\webapps
host=localhost:20000
```

- Use the directory specified by the "file\_path" variable, or modify it to point to your own custom webapp directory. Copy the "struts-documentation.war" and "struts-example.war" files into that webapp directory, and start the UBS (read documentation distributed with UBS for information on how to start it if necessary). Your webapps are now accessible from the following URL:

```
http://localhost/<PLUGIN>/SaServletEngine.class/struts-example/
http://localhost/<PLUGIN>/SaServletEngine.class/struts-documentation/
```

- Note: "<PLUGIN>" represents the plugin you are using for your specific webserver. For Apache on Windows, it might be "cgi-bin/SaCGI.exe", for IIS on Windows, it might be "scripts/SaCGI.exe" or "scripts/ISAPI.dll". Consult the UBS documentation for more information.

- 
- Author: Mike Schachter
- 

Back to [Installation](#)

*Copyright (c) 2000-2002, Apache Software Foundation*

Powered by  
**Struts**



## User Guide

[Table of](#)[Contents](#)[Preface](#)[Introduction](#)[Model](#)[Components](#)[View](#)[Components](#)[Controller](#)[Components](#)[Configuration](#)[Release](#)[Notes](#)[Installation](#)

## Developer Guides

[Bean Tags](#)[HTML](#)[Tags](#)[Logic](#)[Tags](#)[Nested](#)[Tags](#)[Template](#)[Tags](#)[Tiles Tags](#)[Utilities](#)[Validator](#)

## Quick Links

[Welcome](#)[News and](#)[Status](#)[Resources](#)[User and](#)[Developer](#)[Guides \\*](#)[FAQs and](#)[HowTos](#)

## 5.2 Installation

Contributors:

- dIon Gillard

## Installing Struts with your servlet container

### WebSphere Application Server 3.5 FixPack 2

- In the steps below, \$WAS\_HOME refers to the directory in which you have installed WebSphere Application Server, and \$STRUTS\_HOME is the directory in which you unpacked the Struts binary distribution.
- WebSphere before 3.5.2 did not support JSP 1.1 and Servlet 2.2, which made it difficult to get Struts functioning without massive code changes. Please upgrade to 3.5.2 (3.5 with FixPack 2) before attempting to use Struts. See <http://www.ibm.com/software/webervers/appserv/efix.html> for download and install instructions on FixPack 2 for WebSphere 3.5
- Warning: Struts will not work with WebSphere 3.5.2 out of the box. Fixes expected to be in WebSphere 3.5.3 (not released at time of writing) should correct this. However, you can successfully get WebSphere 3.5.2 working with Struts.
- Make sure the WebSphere Application Server is started. Under Windows NT/2000, it's the "IBM WS AdminServer" service.
- Start the WebSphere Administrative Console.
- Once it's started, select "Convert a War File" from the tasks toolbar option, or from the Console->Tasks menu. This will cause a "Convert War File" Wizard dialog to appear.
- Select a Servlet Engine to host the web application that will result from converting the War file, e.g. "Default Servlet Engine", by expanding the tree control under Nodes. Press the Next button.
- Select a Virtual Host to associate the resulting web application with, e.g. "default host". Press the Next button.
- Press the Browse button and choose the \$STRUTS\_HOME/webapps/struts-example.war. Press the Next button
- Select a destination directory for the resulting web application, e.g. \$WAS\_HOME/hosts/default\_host. Press the Next button
- Enter a "Web Application Web Path", e.g. struts-example, and a "Web Application Name", e.g. struts-example. Press the Finish button.
- You should, after a lengthy pause, get a message box with the text Command "convert war file" completed successfully. Press Ok.
- You now need to add jaxp.jar and a jaxp compatible parser, e.g. parser.jar from JAXP 1.0.1 to the struts-example web application's servlets directory, e.g. \$WAS\_HOME/AppServer/hosts/default\_host/struts-example/servlets
- At this point, if WAS 3.5.2 correctly implemented Servlet 2.2, all would be fine. However, WAS 3.5.2 returns null for calls to `ServletContext.getResource(String)` or `ServletContext.getResourceAsStream(String)`. This manifests itself as an exception in the application server stdout log, e.g. default\_server\_stdout.log.
- Warning: Don't be fooled by the fact that the web application starts successfully from the Admin Console. It actually doesn't. The Admin Console is lying. If you try to access the webapp e.g. <http://localhost/struts-example/> it will fail.
- At this point, you need to patch the Struts source. There are three places `getResourceAsStream` is called:

```
ResourceTag.doStartTag()  
ActionServlet.initMapping()  
PropertyMessageResources.loadLocale(String)
```

of these, ActionServlet is the most important.

- Change the source from

```
// Acquire an input stream to our configuration resource  
InputStream input = getServletContext().getResourceAsStream(config);
```

to

```
// Acquire an input stream to our configuration resource  
InputStream input = new  
    java.io.FileInputStream(getServletContext().getRealPath(config));
```

- Make similar changes to the other classes if necessary.
- Recompile ActionServlet and copy the .class file to  
\$WAS\_HOME/AppServer/hosts/default\_host/struts-example/servlets/  
org/apache/struts/action/ActionServlet.class
- Another bug with WAS 3.5.2's classloaders is that Class.getResource() won't load a resource from a  
jar, so you must copy  
\$STRUTS\_HOME/lib/struts-config\_1\_0.dtd  
to  
\$WAS\_HOME/AppServer/hosts/default\_host/struts-  
example/servlets/org/apache/struts/resources/struts-config\_1\_0.dtd  
or be connected to the Internet to fetch the dtd from the Jakarta web site.
- Start your webapp in the Admin Console
- Test the example application by loading the following URL in your browser of choice:  
<http://localhost/struts-example/> > <http://localhost/struts-example/>

- 
- Author: dIon Gillard
- 

Back to [Installation](#)



## User Guide

[Table of Contents](#)

[Preface](#)

[Introduction](#)

[Model](#)

[Components](#)

[View](#)

[Components](#)

[Controller](#)

[Components](#)

[Configuration](#)

[Release Notes](#)

[Installation](#)

## Developer Guides

[Bean Tags](#)

[HTML Tags](#)

[Logic Tags](#)

[Nested Tags](#)

[Template Tags](#)

[Tiles Tags](#)

[Utilities](#)

[Validator](#)

## Quick Links

[Welcome](#)

[News and Status](#)

[Resources](#)

[User and](#)

[Developer](#)

[Guides \\*](#)

[FAQs and](#)

[HowTos](#)

## 5.2 Installation

Contributors:

- Chris Assenza

## Installing Struts with your servlet container

### WebSphere Application Server 3.5 and the Example Application

Server: Windows 2000 Server with WebSphere 3.5.3 Advanced

1. Start up the adminserver.
2. Start up Admin Console.
3. Use the Convert War file task to convert the struts-example.war from the struts-b1 distrib as-is.
4. Convert to the default\_server, default servlet engine and standard install directory (c:\websphere\appserver\hosts\default\_host).
5. Create a WEB-INF directory in the servlets dir and copy struts-config.xml, database.xml AND web.xml into it (Keep WEB-INF with all the TLD's under web - both WEB-INF directories must be present).
6. Copy jaxp 1.0.1's (NOT 1.1.1's) jaxp.jar and parser.jar to the servlets directory of the strut-example webapp.
7. In the servlets directory, open struts.jar with WinZip. Extract the three DTD's (struts-config\_1\_0.dtd, web-app\_2\_2.dtd and web-app\_2\_3.dtd) into the servlets directory making sure you use folder names (so the files extract to servlets/org/apache/struts/resources).
8. Click on struts-example in the Admin Console under Default Server/Default Servlet Engine and click the advanced tab on the right hand side of the screen.
9. Down where it says Default Error Page, enter /ErrorReporter and then click Apply.
10. Start the Default Server via the Admin Console. You should see a whole bunch of ActionServlet messages in the default\_host\_stdout.log file with no exceptions.
11. Via a browser accessed the app using <http://localhost/struts-example/index.jsp>.
12. If it returns "Application not Available" then go back to the Admin Console, right-click on struts-example and select Restart WebApp.
13. Once it reports success, go back to the URL above and try again - it should work flawlessly.

For whatever reason, some installations do not like XML files that reference PUBLIC DTD's - if in looking at the default\_host\_stdout.log file you see errors about invalid public URL references during DTD registrations, or if your pages say "cannot find //login or //saveRegistration (ie. action mappings) then do the following:

1. Stop Default Server
2. Go to servlets\WEB-INF\ and edit web.xml and struts\_config.xml.



3. In the DOCTYPE declaration, change the word PUBLIC to SYSTEM and completely remove the line that reads "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN" from web.xml and remove "-//Apache Software Foundation//DTD Struts Configuration 1.0//EN" from struts-config.xml.
4. Save these changes and go back to step 10 above.

Just as a troubleshooting guide -

If you are getting errors like "Cannot find ActionMappings, etc..." or "Cannot find key org.apache.struts.MESSAGE" then your application is most likely still bombing on the struts-config issue that Richard discovered. The above steps SHOULD correct that leaving nothing out. If you are getting 404 errors about //logon or something not found, then you are still having XML config troubles and it is not initializing the Action servlet properly. Follow the steps above in regards to DTD's and it should work.

As a final thought, I obviously haven't gotten to test too much but I don't believe that there are ANY coding changes that need to be made to the actual struts source. Everything about getting it to work in WebSphere has been a WebSphere configuration issue thus far (and I don't think I'll be having any more).

If changing the DTD's to SYSTEM, do so ONLY AFTER using the Convert a War util. Ant doesn't seem to like it the other way! :)

---

Back to [Installation](#)

---

*Copyright (c) 2000-2002, Apache Software Foundation*

Powered by  
**Struts**



## User Guide

[Table of Contents](#)

[Preface](#)

[Introduction](#)

[Model](#)

[Components](#)

[View](#)

[Components](#)

[Controller](#)

[Components](#)

[Configuration](#)

[Release Notes](#)

[Installation](#)

## Developer Guides

[Bean Tags](#)

[HTML Tags](#)

[Logic Tags](#)

[Nested Tags](#)

[Template Tags](#)

[Tiles Tags](#)

[Utilities](#)

[Validator](#)

## Quick Links

[Welcome](#)

[News and Status](#)

[Resources](#)

[User and  
Developer  
Guides \\*](#)

[FAQs and  
HowTos](#)

## 5.2 Installation

Contributors:

- Robert Hayden
- Wong Kok Kai

## Installing Struts with your servlet container

### Weblogic 5.1 (service pack 8)

- Obtain and install the Xerces XML parser (problems have been reported with the Sun reference implementation). Put xerces.jar in your WebLogic system path.
- Obtain and unpack the Struts binary distribution (this procedure assumes it was extracted to `c:\jakarta-struts`).
- Add an entry to weblogic.properties for each of the Struts web applications that you would like to configure. For example, to make the struts-example application available, add the following line to weblogic.properties:  

```
weblogic.httpd.webApp.strutsexample=
c:/jakarta-struts/webapps/struts-example.war
```
- You do not need to include struts.jar or any of the application specific classes in the WebLogic classpath, since this will be done automatically (unless deploying an unpacked web archive- see below).
- Start WebLogic server and point your web browser to the struts application. For example, to connect to the example application added in step 3:  

```
http://localhost:7001/strutsexample
```
- This example application depends on the Struts specific resource file ApplicationResources.properties to be present on the classpath. However, WebLogic only extracts \*.class files from the archive so this file will not be found, resulting in an error the first time it is needed- something similar to: `javax.servlet.ServletException: runtime failure in custom tag 'message'`. Steps 6 & 7 will need to be performed for this application, and any other that relies on ApplicationResources.properties.
- Extract ApplicationResources.properties from the \*.war file, and manually copy it to the respective package in the \_tmp\_war\_ directory WebLogic created for this application. Again referring to the struts-example application, this would be:  

```
c:\jakarta-struts\webapps\WEB-INF\_tmp_war_strutsexample
```
- Restart WebLogic. You will now be able to run the application:  

```
http://localhost:7001/strutsexample
```

The above steps should be followed for applications deployed as \*.war files. For unpacked web applications, configuration involves adding both `struts.jar` and `/WEB-INF/classes` to the WebLogic classpath. For this reason, I would suggest deploying applications as war files to WebLogic. However, the same example application can be successfully deployed in extracted format by modifying weblogic.properties (assuming the war was extracted to directory `webapps/struts-example`):

```
weblogic.httpd.webApp.strutsexample=
c:/jakarta-struts/webapps/struts-example/
```

And starting WebLogic with the updated WebLogic classpath. For example:

```
c:\jdk1.3\bin\java -ms16m -mx64m
-classpath c:\weblogic\lib\weblogic510sp8boot.jar;
```

---

## Additional Recommendations

- Servlet and JSP-Reloading should be turned off. First, you pay a performance penalty. Depending on the number of JSPs, the number of requests and the configured checking interval, the server will slow down. Second, with JSP- and Servlet reloading, one opens the door for various Weblogic classloader problems, that are difficult to diagnose, difficult to handle and often lead to lost HTTP-sessions.
- Set the name of the `sessionid` to `JSESSIONID`, if cookies are used for session tracking and to `jsessionid`, if sessions are URL-based. (There are additional problems related to URL-based sessions, caused from BEA's way to encode the session id (J2EE-incompatible) as query-param. Especially `<bean:include>` will not work with URL-based sessions yet. However, using the correct session name solves at least some problems.)
- Configure the JSP-Servlet registration in `weblogic.properties` for maximum J2EE-compliance and/or for maximum performance.

The JSP-Servlet supports some initialization parameters that can be customized to get best performance, maximum compliance or (as shown below) easier debugging:

```
weblogic.httpd.initArgs.*.jsp=\
    pageCheckSeconds=-1,\
    setEmptyStrings=false,\
    compileCommand=._jspCompiler_.cmd,\
        workingDir=/weblogic/myserver/tmp_classfiles,\
    keepgenerated=true,\
        debug=true,\
    verbose=true
```

In the above example, the batch file (`_jspCompiler_.cmd`) invokes `jikes` which results dramatically reduced startup times (`jikes` is about three times faster than `javac`.) The batchfile contains only a single line:

```
@jikes -g -nowarn %*
```

The next configuration could be used when all tests have been done and speed is the major concern ...

```
weblogic.httpd.initArgs.*.jsp=\
    pageCheckSeconds=-1,\
    setEmptyStrings=false,\
    compileCommand=._jspCompiler_.cmd,\
        workingDir=/weblogic/myserver/tmp_classfiles,\
    keepgenerated=false,\
        debug=false,\
    verbose=false
```

... together with ...

```
@jikes -O -nowarn %*
```

Weblogic supports similar settings through `<context-params>` in `web.xml` (Please read the latest documentation at the BEA website for details.)

For additional issues, see also [More fixes for WLS 5.1 SP8](#) from the Struts Developer mailing list

- 
- Author: Robert Hayden
  - Author: Wong Kok Kai
  - Author: Matthias Kerkhoff
- 

Back to [Installation](#)

---

*Copyright (c) 2000-2002, Apache Software Foundation*

Powered by  
**Struts**



## User Guide

[Table of Contents](#)[Preface](#)[Introduction](#)[Model](#)[Components](#)[View](#)[Components](#)[Controller](#)[Components](#)[Configuration](#)[Release Notes](#)[Installation](#)

## Developer Guides

[Bean Tags](#)[HTML Tags](#)[Logic Tags](#)[Nested Tags](#)[Template Tags](#)[Tiles Tags](#)[Utilities](#)[Validator](#)

## Quick Links

[Welcome](#)[News and Status](#)[Resources](#)[User and  
Developer  
Guides \\*](#)[FAQs and  
HowTos](#)

## 6.1 Release Notes

Contributors:

- Craig R. McClanahan
- Robert Leland
- Ted Husted
- Martin Cooper

## Beta Notes

This section contains the release notes for **nightly build** of the Struts Framework, for changes that have taken place since [Version 1.1 beta 2](#) was released. For a complete list of changes since the last production release, see the [Introduction](#)

## Beta Fixes

**ApplicationConfig:** In Struts 1.1. beta 3, the ApplicationConfig class is renamed to ModuleConfig, to conform with the venacular.

**Dyna\*Form:** In Struts 1.1. beta 3, the reset method was changed so that it conforms to the original ActionForm implementation.

**Blank application:** Fixed configuration problem with Struts Blank application.

## Introduction

The remainder of this document contains the release notes for **nightly build** of the Struts Framework, and covers changes that have taken place since [Version 1.0.2](#) was released. The following sections cover [New Features](#) and [Changes](#) to Struts.

## What's Included?

The binary distribution of this release includes the following files relevant to Struts:

- **INSTALL** - Brief installation instructions. See the [Struts Documentation Application](#), or online at <http://jakarta.apache.org/struts/> for more information.
- **LICENSE** - The Apache Software Foundation license that defines the terms under which you can use Struts (and other software licensed by Apache).
- **README** - A brief introduction to Struts.
- **lib/** - Directory containing files you will need in your own applications. The individual files of interest are:
  - **commons-\*.jar** - Release packages from the [Jakarta Commons Project](#) that Struts relies on. You are welcome to use these classes in your own applications. These JAR files should be copied into the `/WEB-INF/lib` directory of your web application.
  - **struts.jar** - JAR file that contains the compiled Java classes of Struts. You must place this file in the `/WEB-INF/lib` directory of your web application.
  - **struts-xxxxx.tld** - The tag library descriptor files for the Struts 1.1 tag libraries (bean, html, logic, and template). You must place these files in the `/WEB-INF` directory of your web application, and reference them with appropriate `<taglib>` directives in your web.xml file.
  - **jdbc2\_0-stdext.jar** - The JDBC 2.0 Optional Package API classes (package `javax.sql`). You will need to include this file in the `/WEB-INF/lib` directory of your application, if it is not already made visible to web applications by your servlet container.
  - **struts-config\_1\_1.dtd** - The document type descriptor (DTD) for the Struts 1.1 configuration file (which is typically named `/WEB-INF/struts-config.xml`). Your configuration file will be validated against an internal copy of this DTD -- this copy is available for reference purposes only.
  - **struts-config\_1\_0.dtd** - The document type descriptor (DTD) for the Struts 1.0 configuration file (which is typically named `/WEB-INF/struts-config.xml`). Your configuration file will be validated against an internal copy of this DTD -- this copy is available for reference purposes only.
  - **web-app\_2\_2.dtd** - The document type descriptor (DTD) for web.xml files conforming to the Servlet 2.2 specification. This copy is for reference purposes only.
  - **web-app\_2\_3.dtd** - The document type descriptor (DTD) for web.xml files conforming to the Servlet 2.3 specification. This copy is for reference purposes only.
- **webapps/** - Web Application Archive (WAR) files for the web applications that are included with Struts.

## What's New?

Following are highlights of the new features. In the next section, we provide links to the JavaDocs for the affected classes.

### New Configuration DTD

The Struts Configuration 1.0 DTD has been deprecated in favor of the [struts-config\\_1\\_1.dtd](#). In the Struts 1.1 release, existing Struts configuration files can be loaded using either DTD version.

### New Dependencies on Commons packages

Several components of Struts 1.0 have been found to be useful in general Java development (and not just useful for building Struts-based web applications), and have been migrated into the [Jakarta Commons Project](#). As a result, the current development version of Struts has been modified to rely on the Commons packages containing these classes, rather than the Struts internal versions. In nearly every case, this involved changing only the `import` statements at the top of your classes. Any applications that utilize these classes will need to be modified in the same way.

The following Commons packages contain the replacements for the corresponding Struts 1.0 classes:

- **BeanUtils Package** [[org.apache.commons.beanutils](#)] -  
`org.apache.struts.utils.BeanUtils,`  
`org.apache.struts.utils.ConvertUtils,` and  
`org.apache.struts.utils.PropertyUtils.`
- **Collections Package** [[org.apache.commons.collections](#)] -  
`org.apache.struts.util.ArrayStack,`  
`org.apache.struts.util.FastArrayList,`  
`org.apache.struts.util.FastHashMap,`  
`org.apache.struts.util.FastTreeMap.`
- **Digester Package** - [[org.apache.commons.digester](#)] -  
`org.apache.struts.digester.*.`

The following Commons packages are also now used by various components of the Struts framework:

- **Database Connection Pool Package** [[org.apache.commons.dbpc](#)]
- **FileUpload Package** [[org.apache.commons.fileupload](#)]
- **Logging Package** [[org.apache.commons.logging](#)]
- **Pool Package** [[org.apache.commons.pool](#)]
- **Validator Package** [[org.apache.commons.validator](#)]

### **NOTE! XML Parser Prerequisite Updated**

Struts now depends on an XML parser that conforms to the JAXP/1.1 (rather than JAXP/1.0) APIs. Parsers known to work include the JAXP/1.1 reference implementation, and Xerces 1.3.1.

### **SOURCE DEVELOPERS NOTE! Ant Prerequisite Updated**

To build Struts from source Ant 1.4 or later is now required. This does not affect developers that use Struts from the binary distribution.

### **Struts Validator Integration**

The new Commons-Validator is now integrated with Struts and exposed through the new Validator package.

### **Tiles - An advanced templating taglib**

The Tiles JSP assembly framework has been integrated with Struts.

### **Nested - An very cool taglib extension**

The Nested taglib is bundled with Struts and enhances the functionality of the existing Struts

tags.

## **New Example Applications**

New example applications for the Validator and Tiles are now part of the Struts distribution.

## **New Contrib directory for optional components**

A new directory (`contrib`) in the CVS source repository has been added to accumulate Struts add-on extensions that are generally useful but have not yet been integrated into the standard code base.

- Scaffold - An extension of the Commons Scaffold toolkit of reusable classes for building web applications.
- Struts-el - The optional Struts-el taglib makes it easy to use Struts with JSTL (container with servlet 2.3 support required).

The source for these components is available in the Struts source distribution. Binary distributions may also be made available with the Struts download area. As optional components, these products have their own release cycles.

## **Action Package Additions**

The following new features have been added to the basic controller framework [`org.apache.struts.action`]:

- The `ActionServlet` now provides support for modular Struts applications and sports several new extension points.
- The new `ActionMessages` class will support a superset of the capabilities of `ActionErrors`, and will be useful as a collection of general purpose messages, not just errors.

## **Upload Package Additions**

The following new features have been added to the file upload classes [`org.apache.struts.upload`]:

- `CommonsMultipartRequestHandler`: New class that implements file upload using the Jakarta Commons FileUpload package.

## **Util Package Additions**

The following new features have been added to the utility classes [`org.apache.struts.util`]:

- `LocalStrings`: Correct message regarding replaceable parameter so that it does not append an extraneous character.
- `LabelValueBean`: New class that defines a collection of name/value pairs that can be used with the `<html:options>` tag, and elsewhere.
- `MessageResources`: Escape any single quote characters that are included in the specified message string.
- `computeParameters`: Allow a transaction token to be the only parameter in .
- `RequestUtils`: Change to encode ampersands when building a query string.

## **Bean Taglib Package Additions**



The following new features have been added to the *struts-bean* custom tag library [org.apache.struts.taglib.bean]:

- `<bean:write>`: Add format, locale and bundle attributes to support formatting values according to current user locale, format string from attribute or format string from string resources.
- `<bean:cookie>`, `<bean:header>`, or `<bean:parameter>`: Correct the generated scripting variable type when tag is used with the "multiple" attribute.
- `<bean:message>`: Added name, property, and scope attributes to the tag, so that the message source key can be obtained dynamically from a bean or bean property.

### HTML Taglib Package Additions

The following new features have been added to the *struts-html* custom tag library [org.apache.struts.taglib.html]:

- `<html:link>`: Added 'action' attribute.
- `<html:options>`: If the property specified by the 'property' attribute returns null, tag now throws an error message that indicates what the real problem is, rather than causing an NPE.
- `<html:option>` and `<html:options>`: Added 'style' and 'styleClass' attributes.
- `<html:optionsCollection>`: New tag providing a cleaner way of populating HTML options from a collection.
- `<bean:message>`: Added 'name', 'property' and 'scope' attributes so that the message resource key can be obtained dynamically from a bean.
- `<html:messages>`: New tag to iterate through a message collection in the new ActionMessages class.
- `ActionForm`: Tag will now call `reset()` if it instantiates the ActionForm bean. This also requires that the bean instantiated by the tag to be an ActionForm subclass.
- `<html:image>`: Added the 'align' attribute.
- `<html:img>`: Added the mouse event attributes ('onclick', 'ondblclick', 'onmousedown', 'onmouseup', 'onmouseover', 'onmousemove', 'onmouseout').
- `SubmitTag`, `SelectTag`, `LinkTag.java`, `CheckboxTag`, `ButtonTag`, `ImageTag`, `RadioTag`, and `TextArea` tags: Added indexed property.

### Logic Taglib Package Additions

The following new features have been added to the *struts-logic* custom tag library [org.apache.struts.taglib.logic]:

- `<logic:empty>` and `<logic:notEmpty>`: New tags that are similar to `<logic:present>` and `<logic:notPresent>` except for the treatment of empty strings.

### Template Taglib Package Additions

The following new features have been added to the *struts-template* custom tag library [org.apache.struts.taglib.template]:

- None.

### Documentation Additions

The following new features have been added to the Struts Documentation application (and corresponding contents on the Struts web site):

- Version Differences: New section in Release Notes to link to the JavaDocs for all Struts classes and members added or changed between versions.
- FAQ/HowTos: New documentation category organizes our FAQs and example-driven howTos. New HowTos include "Building Applications", "Using SSL", and using Struts with Eclipse or NetBeans.
- User Guide Preface: New section to overview the enabling technologies behind Struts.
- Developer Guides: Added "cover page" to guides. These then link to the Package Descriptions and the API guides.
- HTML tag documentation: expanded to cover using indexed properties with iterate.
- Site Menu: Removed separate links to taglib documentation, since these are now in the Developer Guide.
- Newbie FAQ: The questions most likely to be asked by new developers using Struts. Still under development.
- Kickstart FAQ: The questions most likely to be asked when selecting Struts.
- 1.0 JavaDoc: Added archival copy to web site for future reference.
- The UserGuide "Building" pages: General revisions to reflect new features and current practices.
- Installation: Updated instructions for SilverStream and Resin. Add installation notes for Jetty. Added RexIP to list of nominal containers.
- JavaDocs: New @since Struts 1.1 tag to indicate new packages, classes, and members added after the Struts 1.0.x version

## Operational Changes and Bug Fixes

### Struts Configuration Changes

The following changes and bug fixes have occurred in the configuration files related to Struts:

- Deprecated (Struts 0.5) configuration file format: Remove support.
- Deprecated (Struts 0.5) methods: Remove from codebase.

### Added Config Package

- ControllerConfig: Added inputForward property to indicate that ActionMapping.input is a forward rather than a URI.
- ControllerConfig: Added forwardPattern and inputPattern to help manage page directories in application modules.
- Added package to provide more flexibility in configuring the controller and to provide support for modular applications

### Action Package Changes

The following changes and bug fixes have occurred in the basic controller framework (package `org.apache.struts.action`):

- ActionMapping: input property may now refer to an ActionForward rather than a module-relative path if inputForward is set to true on the module's ControllerConfig bean [`org.apache.struts.config.ControllerConfig`].
- ActionServlet: Added convertNull parameter to simulate the Struts 1.0 behavior when populating forms. If set to true, the numeric Java wrapper class types (like `java.lang.Integer`) will default to null (rather than 0).

- **ActionServlet:** Added "config/\$foo" parameter and deprecated several others in favor of components in the new config package.
- **ActionForms and related classes:** now use a StringBuffer when responding a toString request in order to conserve resources.
- **LookupDispatchAction:** Added standard Action to help select between internationalized buttons.
- **ActionForm class:** Modified to use ActionServletWrapper rather than expose ActionServlet.
- **ActionServletWrapper class:** Added for use by ActionForm to prevent the Public String properties of ActionServlet from being changed via a query string.
- **Action.MAPPING\_KEY request attribute:** Unconditionally pass the selected mapping as a request attribute ("org.apache.struts.action.mapping.instance") even if no form bean is specified.
- **ActionServlet:** Avoid a NullPointerException in corner cases caused by failed initialization of servlet.
- **ActionForm class:** Now truly serializable, because the two non-serializable instance variables (servlet and multipartRequestHandler) have been made transient. However, if you actually do serialize and deserialize such instances, it is your responsibility to reset these two properties.
- **ActionMessages and ActionErrors:** The initial order a property/key is added in is now retained.

### Upload Package Changes

The following changes and bug fixes have occurred in the file upload package (package `org.apache.struts.upload`):

- **CommonsMultipartRequestHandler:** New implementation of file upload based on the Jakarta Commons FileUpload package. This new implementation is now the default.
- **BufferedMultipartInputStream:** Fixed lost byte problem.
- **ArrayIndexOutOfBoundsException:** Fixed situations where this was known to occur.
- **Multipart requests:** Better reporting for premature closing of input streams while reading multipart requests.
- **New line characters:** Additional fix for file corruption problem with uploads and new line characters.

### Utility Package Changes

The following changes and bug fixes have occurred in the utilities (package `org.apache.struts.util`):

- **RequestUtils:** Added support for forwardPattern, pagePattern, and inputForward properties on ControllerConfig.
- **GenericDataSource:** Deprecated and modified to act as a thin wrapper around `[org.apache.commons.dbpc.BasicDataSource]`. Replaced by direct use of BasicDataSource or other compatible component.
- **RequestUtils class:** Modify to use ActionServletWrapper rather than expose ActionServlet.
- **Added error message for the getActionErrors and getActionMessages method.**
- **getActionErrors and getActionMessages:** Added methods to generate the correct corresponding object based on the object retrieved from request scope based on the key passed in.
- **ActionErrors or ActionMessages:** The logic for creating one of these objects has been moved to a utility method in RequestUtils.
- **JspException message:** Now generated in RequestUtils.
- **ConvertUtils.convertCharacter():** Will now detect empty strings and return the default

value.

### Bean Taglib Package Changes

The following changes and bug fixes have occurred in the *struts-bean* custom tag library (package `org.apache.struts.taglib.bean`):

- `<html:errors>`: When the property tag is specified, errors are no longer printed if the specified property has no errors. Previously errors were always printed ! Future enhancements would include additional attributes to always turn off the header or footer.
- Made the remaining helper methods "protected" rather than "private".

### HTML Taglib Package Changes

The following changes and bug fixes have occurred in the *struts-html* custom tag library (package `org.apache.struts.taglib.html`):

- `FormTag`: Fixed to exclude query string when identifying action mapping name.
- `ImgTag`: Correctly URLEncode the query string parameter value even if there is only a single parameter.
- `MultiboxTag.doAfterBody()`: Corrected to return `SKIP_BODY` instead of `SKIP_PAGE`.

### Logic Taglib Package Changes

The following changes and bug fixes have occurred in the *struts-logic* custom tag library (package `org.apache.struts.taglib.logic`):

- None.

### Documentation Application Changes

The following changes and bug fixes to the Struts Documentation application (and corresponding contents on the Struts web site) have occurred:

- Reorganized Resources into separate pages..
- In the Tag Developers Guide, add more detail regarding file upload requirements.
- In Building View Components, clarify that additional i18n support may be provided by the browser, and is outside the scope of the framework.
- In Building Controller Components, document 'validating' init-param, add defaults for various parameters, clarify that some web.xml settings are not Struts-specific.
- Tag library documentation: Moved under User's Guide.
- Reorganized to separate 1.0 material from nightly build material.

### MailReader Example Application Changes

The following changes and bug fixes to the Struts MailReader Example Application have occurred:

- Add Russian and Japanese translations of the application resources and set the character set for the example JSP pages to "UTF-8" so that it can display either English or Japanese.
- Exchange "name" for "attribute" properties for Edit mappings in Struts configuration file.
- Remove references to saving database data from "tour" document, since this functionality was removed.

## Template Example Application Changes

The following changes and bug fixes to the Struts Template Example Application have occurred:

- None.

## Exercise Taglib Example Application Changes

The following changes and bug fixes to the Struts Exercise Taglib Example Application have occurred:

- Added test case for `<html:link>` using "action" attribute.
- Added test case for `<html:select>` using `<html:options>` based on a collection saved in the page context.

## What's different?

This section provides links to the Struts JavaDoc for any classes that have been added or deprecated since the Struts 1.0 release.

### Previously deprecated classes and packages removed in Struts 1.1

- Removed: `org.apache.struts.utils.BeanUtils`, `org.apache.struts.utils.ConvertUtils`, and `org.apache.struts.utils.PropertyUtils` - replaced by [org.apache.commons.beanutils](#)
- Removed: `org.apache.struts.util.ArrayStack`, `org.apache.struts.util.FastArrayList`, `org.apache.struts.util.FastHashMap`, `org.apache.struts.util.FastTreeMap` - replaced by [org.apache.commons.collections](#)
- Removed: `org.apache.struts.digester.*` - replaced by [org.apache.commons.digester](#)
- Removed: The `struts-config.dtd` - Replaced by [struts-config 1.1.dtd](#).
- Removed: The omnibus "struts" taglib and its associated TLD - replaced by separate bean, logic, and html taglibs.
- Removed: The "form" taglib and its associated TLD - replaced by (renamed as) the html taglib.

### Packages added in Struts 1.1

- [config](#)
- [taglib.nested](#)
- [taglib.nested.bean](#)
- [taglib.nested.html](#)
- [taglib.nested.logic](#)
- [validator](#)

### Classes added in Struts 1.1

action

- [ActionMessage](#)
- [ActionMessages](#)
- [DynaActionForm](#)
- [DynaActionFormClass](#)
- [ExceptionHandler](#)
- [RequestProcessor](#)

actions

- [LookupDispatchAction](#)

taglib.html

- [FrameTag](#)
- [JavascriptValidatorTag](#)
- [MessagesTag](#)
- [OptionsCollectionTag](#)

taglib.logic

- [EmptyTag](#)
- [MessagesNotPresentTag](#)
- [MessagesPresentTag](#)

upload

- [CommonsMultipartRequestHandler](#)

util

- [LabelValueBean](#)

## **Classes with members added in Struts 1.1**

### [action.Action](#)

- ACTION\_SERVLET\_KEY
- APPLICATION\_KEY
- MESSAGE\_KEY
- PLUG\_INS\_KEY
- REQUEST\_PROCESSOR\_KEY
- execute
- getResources(javax.servlet.http.HttpServletRequest)
- saveMessages

### [action.ActionServlet](#)

- configDigester
- convertHack
- log
- processor
- getInternal

- `destroyApplications`
- `destroyConfigDigester`
- `getApplicationConfig`
- `getRequestProcessor`
- `initApplicationConfig`
- `initApplicationDataSources`
- `initApplicationPlugIns`
- `initApplicationMessageResources`
- `initConfigDigester`
- methods created for backward-compatibility only
  - `defaultControllerConfig`
  - `defaultFormBeansConfig`
  - `defaultForwardsConfig`
  - `defaultMappingsConfig`
  - `defaultMessageResourcesConfig`

#### [taglib.html.BaseHandlerTag](#)

- `indexed`
- `setIndexed`
- `getIndexed`

### **Classes deprecated between Struts 1.0 and Struts 1.1**

action

- [ActionException](#)
- [ActionFormBeans](#)
- [ActionForwards](#)
- [ActionMappings](#)

### **Classes with members deprecated between Struts 1.0 and Struts 1.1**

#### [action.Action](#)

- `FORM_BEANS_KEY`
- `FORWARDS_KEY`
- `MAPPINGS_KEY`
- `getResources()`
- `perform`

#### [ActionServlet](#)

- `findDataSource`
- `findFormBean`
- `findForward`
- `findMapping`
- `initDataSources`
- methods created for backward-compatibility only
  - `defaultControllerConfig`
  - `defaultFormBeansConfig`
  - `defaultForwardsConfig`
  - `defaultMappingsConfig`
  - `defaultMessageResourcesConfig`

Next: [Installation](#)

---

*Copyright (c) 2000-2002, Apache Software Foundation*

**Powered by**  
**Struts**





## User Guide

[Table of Contents](#)[Preface](#)[Introduction](#)[Model](#)[Components](#)[View](#)[Components](#)[Controller](#)[Components](#)[Configuration](#)[Release Notes](#)[Installation](#)

## Developer Guides

[Bean Tags](#)[HTML Tags](#)[Logic Tags](#)[Nested Tags](#)[Template Tags](#)[Tiles Tags](#)[Utilities](#)[Validator](#)

## Quick Links

[Welcome](#)[News and Status](#)[Resources](#)[User and  
Developer  
Guides \\*](#)[FAQs and  
HowTos](#)

## Introduction

This document contains the release notes for **Version 1.1 Beta 2** of the Struts Framework, and covers changes that have taken place since [Version 1.0.2](#) was released. The following sections cover [New Features](#) and [Changes](#) to Struts.

## What's Included?

The binary distribution of this release includes the following files relevant to Struts:

- **INSTALL** - Brief installation instructions. See the Struts Documentation Application, or online at <http://jakarta.apache.org/struts/> for more information.
- **LICENSE** - The Apache Software Foundation license that defines the terms under which you can use Struts (and other software licensed by Apache).
- **README** - A brief introduction to Struts.
- **lib/** - Directory containing files you will need in your own applications. The individual files of interest are:
  - **commons-\*.jar** - Release packages from the [Jakarta Commons Project](#) that Struts relies on. You are welcome to use these classes in your own applications. These JAR files should be copied into the `/WEB-INF/lib` directory of your web application.
  - **struts.jar** - JAR file that contains the compiled Java classes of Struts. You must place this file in the `/WEB-INF/lib` directory of your web application.
  - **struts-xxxxx.tld** - The tag library descriptor files for the Struts 1.1 tag libraries (bean, html, logic, and template). You must place these files in the `/WEB-INF` directory of your web application, and reference them with appropriate `<taglib>` directives in your web.xml file.
  - **jdbc2\_0-stdext.jar** - The JDBC 2.0 Optional Package API classes (package `javax.sql`). You will need to include this file in the `/WEB-INF/lib` directory of your application, if it is not already made visible to web applications by your servlet container.
  - **struts-config\_1\_1.dtd** - The document type descriptor (DTD) for the Struts 1.1 configuration file (which is typically named `/WEB-INF/struts-config.xml`). Your configuration file will be validated against an internal copy of this DTD -- this copy is available for reference purposes only.
  - **struts-config\_1\_0.dtd** - The document type descriptor (DTD) for the Struts 1.0 configuration file (which is typically named `/WEB-INF/struts-config.xml`). Your configuration file will be validated against an internal copy of this DTD -- this copy is available for reference purposes only.
  - **web-app\_2\_2.dtd** - The document type descriptor (DTD) for web.xml files conforming to the Servlet 2.2 specification. This copy is for reference purposes only.
  - **web-app\_2\_3.dtd** - The document type descriptor (DTD) for web.xml files conforming to the Servlet 2.3 specification. This copy is for reference purposes only.
- **webapps/** - Web Application Archive (WAR) files for the web applications that are included with Struts.

## What's New?

Following are highlights of the new features. In the next section, we provide links to the JavaDocs for the affected classes.

### New Configuration DTD

The Struts Configuration 1.0 DTD has been deprecated in favor of the [struts-config\\_1\\_1.dtd](#). In the Struts 1.1 release, existing Struts configuration files can be loaded using either DTD version.

### New Dependencies on Commons packages

Several components of Struts 1.0 have been found to be useful in general Java development (and not just useful for building Struts-based web applications), and have been migrated into the [Jakarta Commons Project](#). As a result, the current development version of Struts has been modified to rely on the Commons packages containing these classes, rather than the Struts internal versions. In nearly every case, this involved changing only the `import` statements at the top of your classes. Any applications that utilize these classes will need to be modified in the same way.

The following Commons packages contain the replacements for the corresponding Struts 1.0 classes:

- **BeanUtils Package** [[org.apache.commons.beanutils](#)] -  
`org.apache.struts.utils.BeanUtils`,  
`org.apache.struts.utils.ConvertUtils`, and  
`org.apache.struts.utils.PropertyUtils`.
- **Collections Package** [[org.apache.commons.collections](#)] -  
`org.apache.struts.util.ArrayStack`,  
`org.apache.struts.util.FastArrayList`,  
`org.apache.struts.util.FastHashMap`,  
`org.apache.struts.util.FastTreeMap`.
- **Digester Package** - [[org.apache.commons.digester](#)] -  
`org.apache.struts.digester.*`.

The following Commons packages are also now used by various components of the Struts framework:

- **Database Connection Pool Package** [[org.apache.commons.dbcp](#)]
- **FileUpload Package** [[org.apache.commons.fileupload](#)]
- **Logging Package** [[org.apache.commons.logging](#)]
- **Pool Package** [[org.apache.commons.pool](#)]
- **Services Package** [[org.apache.commons.services](#)]
- **Validator Package** [[org.apache.commons.validator](#)]

### NOTE! XML Parser Prerequisite Updated

Struts now depends on an XML parser that conforms to the JAXP/1.1 (rather than JAXP/1.0) APIs. Parsers known to work include the JAXP/1.1 reference implementation, and Xerces 1.3.1.

## **SOURCE DEVELOPERS NOTE! Ant Prerequisite Updated**

To build Struts from source Ant 1.4 or later is now required. This does not affect developers that use Struts from the binary distribution.

## **Struts Validator Integration**

The new Commons-Validator is now integrated with Struts and exposed through the new Validator package.

## **Tiles - An advanced templating taglib**

The Tiles JSP assembly framework has been integrated with Struts.

## **Nested - An very cool taglib extension**

The Nested taglib is bundled with Struts and enhances the functionality of the existing Struts tags.

## **New Example Applications**

New example applications for the Validator and Tiles are now part of the Struts distribution.

## **New Contrib directory for optional components**

A new directory (`contrib`) in the CVS source repository has been added to accumulate Struts add-on extensions that are generally useful but have not yet been integrated into the standard code base.

- Scaffold - Library of reusable classes for building web applications.
- Service Manager - Add custom services without subclassing controller.

The source for these components is available in the Struts source distribution. Binary distributions may also be made available with the Struts download area. As optional components, these products have their own release cycles.

## **Action Package Additions**

The following new features have been added to the basic controller framework [`org.apache.struts.action`]:

- The `ActionServlet` now provides support for modular Struts applications and sports several new extension points.
- The new `ActionMessages` class will support a superset of the capabilities of `ActionErrors`, and will be useful as a collection of general purpose messages, not just errors.

## **Upload Package Additions**

The following new features have been added to the file upload classes [`org.apache.struts.upload`]:

- `CommonsMultipartRequestHandler`: New class that implements file upload using the Jakarta Commons FileUpload package.

## Util Package Additions

The following new features have been added to the utility classes  
[org.apache.struts.util]:

- `LocalStrings`: Correct message regarding replaceable parameter so that it does not append an extraneous character.
- `LabelValueBean`: New class that defines a collection of name/value pairs that can be used with the `<html:options>` tag, and elsewhere.
- `MessageResources`: Escape any single quote characters that are included in the specified message string.
- `computeParameters`: Allow a transaction token to be the only parameter in .
- `RequestUtils`: Change to encode ampersands when building a query string.

## Bean Taglib Package Additions

The following new features have been added to the *struts-bean* custom tag library  
[org.apache.struts.taglib.bean]:

- `<bean:write>`: Add format, locale and bundle attributes to support formatting values according to current user locale, format string from attribute or format string from string resources.
- `<bean:cookie>`, `<bean:header>`, or `<bean:parameter>`: Correct the generated scripting variable type when tag is used with the "multiple" attribute.
- `<bean:message>`: Added name, property, and scope attributes to the tag, so that the message source key can be obtained dynamically from a bean or bean property.

## HTML Taglib Package Additions

The following new features have been added to the *struts-html* custom tag library  
[org.apache.struts.taglib.html]:

- `<options>`: If the property specified by the 'property' attribute returns null, tag now throws an error message that indicates what the real problem is, rather than causing an NPE.
- `<html:option>` and `<html:options>`: Added 'style' and 'styleClass' attributes.
- `<html:optionsCollection>`: New tag providing a cleaner way of populating HTML options from a collection.
- `<bean:message>`: Added 'name', 'property' and 'scope' attributes so that the message resource key can be obtained dynamically from a bean.
- `<html:messages>`: New tag to iterate through a message collection in the new `ActionMessages` class.
- `ActionForm`: Tag will now call `reset()` if it instantiates the `ActionForm` bean. This also requires that the bean instantiated by the tag to be an `ActionForm` subclass.
- `<html:image>`: Added the 'align' attribute.
- `<html:img>`: Added the mouse event attributes ('onclick', 'ondblclick', 'onmousedown', 'onmouseup', 'onmouseover', 'onmousemove', 'onmouseout').
- `SubmitTag`, `SelectTag`, `LinkTag.java`, `CheckboxTag`, `ButtonTag`, `ImageTag`, `RadioTag`, and `TextArea` tags: Added indexed property.

## Logic Taglib Package Additions

The following new features have been added to the *struts-logic* custom tag library  
[org.apache.struts.taglib.logic]:

- `<logic:empty>` and `<logic:notEmpty>`: New tags that are similar to `<logic:present>` and `<logic:notPresent>` except for the treatment of empty strings.

### Template Taglib Package Additions

The following new features have been added to the *struts-template* custom tag library [`org.apache.struts.taglib.template`]:

- None.

### Documentation Additions

The following new features have been added to the Struts Documentation application (and corresponding contents on the Struts web site):

- Version Differences: New section in Release Notes to link to the JavaDocs for all Struts classes and members added or changed between versions.
- User Guide Preface: New section to overview the enabling technologies behind Struts.
- Developer Guides: Added "cover page" to guides. These then link to the Package Descriptions and the API guides.
- HTML tag documentation: expanded to cover using indexed properties with iterate.
- Site Menu: Removed separate links to taglib documentation, since these are now in the Developer Guide.
- Newbie FAQ: The questions most likely to be asked by new developers using Struts. Still under development.
- Kickstart FAQ: The questions most likely to be asked when selecting Struts.
- 1.0 JavaDoc: Added archival copy to web site for future reference.
- The UserGuide "Building" pages: General revisions to reflect new features and current practices.
- Installation: Updated instructions for SilverStream and Resin. Add installation notes for Jetty. Added RexIP to list of nominal containers.
- JavaDocs: New `@since Struts 1.1` tag to indicate new packages, classes, and members added after the Struts 1.0.x version

## Operational Changes and Bug Fixes

### Struts Configuration Changes

The following changes and bug fixes have occurred in the configuration files related to Struts:

- Deprecated (Struts 0.5) configuration file format: Remove support.
- Deprecated (Struts 0.5) methods: Remove from codebase.

### Added Config Package

- ControllerConfig: Added `inputForward` property to indicate that `ActionMapping.input` is a forward rather than a URI.
- ControllerConfig: Added `forwardPattern` and `inputPattern` to help manage page directories in application modules.
- Added package to provide more flexibility in configuring the controller and to provide support for modular applications

## Action Package Changes

The following changes and bug fixes have occurred in the basic controller framework (package `org.apache.struts.action`):

- **ActionMapping:** input property may now refer to an `ActionForward` rather than a module-relative path if `inputForward` is set to true on the module's `ControllerConfig` bean [`org.apache.struts.config.ControllerConfig`].
- **ActionServlet:** Added `convertNull` parameter to simulate the Struts 1.0 behavior when populating forms. If set to true, the numeric Java wrapper class types (like `java.lang.Integer`) will default to null (rather than 0).
- **ActionServlet:** Added "`config/$foo`" parameter and deprecated several others in favor of components in the new config package.
- **ActionForms** and related classes: now use a `StringBuffer` when responding a `toString` request in order to conserve resources.
- **LookupDispatchAction:** Added standard Action to help select between internationalized buttons.
- **ActionForm** class: Modified to use `ActionServletWrapper` rather than expose `ActionServlet`.
- **ActionServletWrapper** class: Added for use by `ActionForm` to prevent the Public String properties of `ActionServlet` from being changed via a query string.
- **Action.MAPPING\_KEY** request attribute: Unconditionally pass the selected mapping as a request attribute ("`org.apache.struts.action.mapping.instance`") even if no form bean is specified.
- **ActionServlet:** Avoid a `NullPointerException` in corner cases caused by failed initialization of servlet.
- **ActionForm** class: Now truly serializable, because the two non-serializable instance variables (`servlet` and `multipartRequestHandler`) have been made transient. However, if you actually do serialize and deserialize such instances, it is your responsibility to reset these two properties.
- **ActionMessages** and **ActionErrors:** The initial order a property/key is added in is now retained.

## Upload Package Changes

The following changes and bug fixes have occurred in the file upload package (package `org.apache.struts.upload`):

- **CommonsMultipartRequestHandler:** New implementation of file upload based on the Jakarta Commons FileUpload package. This new implementation is now the default.
- **BufferedMultipartInputStream:** Fixed lost byte problem.
- **ArrayIndexOutOfBoundsException:** Fixed situations where this was known to occur.
- **Multipart requests:** Better reporting for premature closing of input streams while reading multipart requests.
- **New line characters:** Additional fix for file corruption problem with uploads and new line characters.

## Utility Package Changes

The following changes and bug fixes have occurred in the utilities (package `org.apache.struts.util`):

- **RequestUtils:** Added support for `forwardPattern`, `pagePattern`, and `inputForward` properties on `ControllerConfig`.
- **GenericDataSource:** Deprecated and modified to act as a thin wrapper around [`org.apache.commons.dbpc.BasicDataSource`]. Replaced by direct use of

BasicDataSource or other compatible component.

- RequestUtils class: Modify to use ActionServletWrapper rather than expose ActionServlet.
- Added error message for the getActionErrors and getActionMessages method.
- getActionErrors and getActionMessages: Added methods to generate the correct corresponding object based on the object retrieved from request scope based on the key passed in.
- ActionErrors or ActionMessages: The logic for creating one of these objects has been moved to a utility method in RequestUtils.
- JspException message: Now generated in RequestUtils.
- ConvertUtils.convertCharacter(): Will now detect empty strings and return the default value.

### Bean Taglib Package Changes

The following changes and bug fixes have occurred in the *struts-bean* custom tag library (`org.apache.struts.taglib.bean`):

- `<html:errors>`: When the property tag is specified, errors are no longer printed if the specified property has no errors. Previously errors were always printed ! Future enhancements would include additional attributes to always turn off the header or footer.
- Made the remaining helper methods "protected" rather than "private".

### HTML Taglib Package Changes

The following changes and bug fixes have occurred in the *struts-html* custom tag library (`package org.apache.struts.taglib.html`):

- FormTag: Fixed to exclude query string when identifying action mapping name.
- ImgTag: Correctly URLEncode the query string parameter value even if there is only a single parameter.
- MultiboxTag.doAfterBody(): Corrected to return SKIP\_BODY instead of SKIP\_PAGE.

### Logic Taglib Package Changes

The following changes and bug fixes have occurred in the *struts-logic* custom tag library (`package org.apache.struts.taglib.logic`):

- None.

### Documentation Application Changes

The following changes and bug fixes to the Struts Documentation application (and corresponding contents on the Struts web site) have occurred:

- Reorganized Resources into separate pages..
- In the Tag Developers Guide, add more detail regarding file upload requirements.
- In Building View Components, clarify that additional i18n support may be provided by the browser, and is outside the scope of the framework.
- In Building Controller Components, document 'validating' init-param, add defaults for various parameters, clarify that some web.xml settings are not Struts-specific.
- Tag library documentation: Moved under User's Guide.
- Reorganized to separate 1.0 material from nightly build material.

### MailReader Example Application Changes

The following changes and bug fixes to the Struts MailReader Example Application have occurred:

- Add Russian and Japanese translations of the application resources and set the character set for the example JSP pages to "UTF-8" so that it can display either English or Japanese.
- Exchange "name" for "attribute" properties for Edit mappings in Struts configuration file.
- Remove references to saving database data from "tour" document, since this functionality was removed.

### Template Example Application Changes

The following changes and bug fixes to the Struts Template Example Application have occurred:

- None.

### Exercise Taglib Example Application Changes

The following changes and bug fixes to the Struts Exercise Taglib Example Application have occurred:

- Added test case for `<html:select>` using `<html:options>` based on a collection saved in the page context.

## What's different?

This section provides links to the Struts JavaDoc for any classes that have been added or deprecated since the Struts 1.0 release.

### Previously deprecated classes and packages removed in Struts 1.1

- Removed: `org.apache.struts.utils.BeanUtils`, `org.apache.struts.utils.ConvertUtils`, and `org.apache.struts.utils.PropertyUtils` - replaced by [org.apache.commons.beanutils](#)
- Removed: `org.apache.struts.util.ArrayStack`, `org.apache.struts.util.FastArrayList`, `org.apache.struts.util.FastHashMap`, `org.apache.struts.util.FastTreeMap` - replaced by [org.apache.commons.collections](#)
- Removed: `org.apache.struts.digester.*` - replaced by [org.apache.commons.digester](#)
- Removed: The `struts-config.dtd` - Replaced by [struts-config 1 1.dtd](#).
- Removed: The omnibus "struts" taglib and its associated TLD - replaced by separate bean, logic, and html taglibs.
- Removed: The "form" taglib and its associated TLD - replaced by (renamed as) the html taglib.

### Packages added in Struts 1.1

- [config](#)



- [taglib.nested](#)
- [taglib.nested.bean](#)
- [taglib.nested.html](#)
- [taglib.nested.logic](#)
- [validator](#)

## Classes added in Struts 1.1

### action

- [ActionMessage](#)
- [ActionMessages](#)
- [DynaActionForm](#)
- [DynaActionFormClass](#)
- [ExceptionHandler](#)
- [RequestProcessor](#)

### actions

- [LookupDispatchAction](#)

### taglib.html

- [FrameTag](#)
- [JavascriptValidatorTag](#)
- [MessagesTag](#)
- [OptionsCollectionTag](#)

### taglib.logic

- [EmptyTag](#)
- [MessagesNotPresentTag](#)
- [MessagesPresentTag](#)

### upload

- [CommonsMultipartRequestHandler](#)

### util

- [LabelValueBean](#)

## Classes with members added in Struts 1.1

### [action.Action](#)

- ACTION\_SERVLET\_KEY
- APPLICATION\_KEY
- MESSAGE\_KEY
- PLUG\_INS\_KEY
- REQUEST\_PROCESSOR\_KEY
- execute

- getResources(javax.servlet.http.HttpServletRequest)
- saveMessages

#### [action.ActionServlet](#)

- configDigester
- convertHack
- log
- processor
- getInternal
- destroyApplications
- destroyConfigDigester
- getApplicationConfig
- getRequestProcessor
- initApplicationConfig
- initApplicationDataSources
- initApplicationPlugIns
- initApplicationMessageResources
- initConfigDigester
- methods created for backward-compatibility only
  - defaultControllerConfig
  - defaultFormBeansConfig
  - defaultForwardsConfig
  - defaultMappingsConfig
  - defaultMessageResourcesConfig

#### [taglib.html.BaseHandlerTag](#)

- indexed
- setIndexed
- getIndexed

### **Classes deprecated between Struts 1.0 and Struts 1.1**

#### action

- [ActionException](#)
- [ActionFormBeans](#)
- [ActionForwards](#)
- [ActionMappings](#)

### **Classes with members deprecated between Struts 1.0 and Struts 1.1**

#### [action.Action](#)

- FORM\_BEANS\_KEY
- FORWARDS\_KEY
- MAPPINGS\_KEY
- getResources()
- perform

#### [ActionServlet](#)

- findDataSource

- findFormBean
- findForward
- findMapping
- initDataSources
- methods created for backward-compatibility only
  - defaultControllerConfig
  - defaultFormBeansConfig
  - defaultForwardsConfig
  - defaultMappingsConfig
  - defaultMessageResourcesConfig

Next: [Installation](#)

## Known Issues

The following known issues will be addressed before Final Release of Struts 1.1. The numbers in parentheses are for the corresponding entries in the [Bugzilla](#) bug tracking system.

### Custom Tags

- The `<html:form>` tag generates incorrect JavaScript when the focus is to be set on a radio button. ([#1586](#))

### Documentation

- Some parts of the documentation are incomplete, especially for new features in this release. These sections are marked TODO. ([#10537](#))

### Example Webapps

- The *struts-blank* sample webapp fails to run due to problems accessing message resources. ([#10955](#))

### Validator Framework

- Validator generates incorrect JavaScript for the `<select>` element. ([#7353](#))
- Validator range checking supports only integer values, and not other numeric values such as floats and doubles. ([#10191](#))
- Validator does not work with Struts application modules other than the default module. ([#10348](#))
- Validator does not successfully validate empty date fields. ([#10349](#))
- Validator does not validate data when using `DynaValidatorActionForm`. ([#10432](#))
- Validator plugin does not correctly configure multiple validation files. ([#10584](#))

Next: [Installation](#)



## User Guide

[Table of Contents](#)[Preface](#)[Introduction](#)[Model](#)[Components](#)[View](#)[Components](#)[Controller](#)[Components](#)[Configuration](#)[Release Notes](#)[Installation](#)

## Developer Guides

[Bean Tags](#)[HTML Tags](#)[Logic Tags](#)[Nested Tags](#)[Template Tags](#)[Tiles Tags](#)[Utilities](#)[Validator](#)

## Quick Links

[Welcome](#)[News and Status](#)[Resources](#)[User and  
Developer  
Guides \\*](#)[FAQs and  
HowTos](#)

## Introduction

This document contains the release notes for **Version 1.1 Beta 1** of the Struts Framework, and covers changes that have taken place since [Version 1.0.1](#) was released. The following sections cover [New Features](#) and [Changes](#) to Struts.

## What's Included?

The binary distribution of this release includes the following files relevant to Struts:

- **INSTALL** - Brief installation instructions. See the Struts Documentation Application, or online at <http://jakarta.apache.org/struts/> for more information.
- **LICENSE** - The Apache Software Foundation license that defines the terms under which you can use Struts (and other software licensed by Apache).
- **README** - A brief introduction to Struts.
- **lib/** - Directory containing files you will need in your own applications. The individual files of interest are:
  - **commons-\*.jar** - Release packages from the [Jakarta Commons Project](#) that Struts relies on. You are welcome to use these classes in your own applications. These JAR files should be copied into the `/WEB-INF/lib` directory of your web application.
  - **struts.jar** - JAR file that contains the compiled Java classes of Struts. You must place this file in the `/WEB-INF/lib` directory of your web application.
  - **struts-xxxxx.tld** - The tag library descriptor files for the Struts 1.0 tag libraries (bean, html, logic, and template). You must place these files in the `/WEB-INF` directory of your web application, and reference them with appropriate `<taglib>` directives in your web.xml file. **NOTE** - The struts-form.tld file is deprecated; you should use the struts-html.tld file instead.
  - **jdbc2\_0-stdext.jar** - The JDBC 2.0 Optional Package API classes (package `javax.sql`). You will need to include this file in the `/WEB-INF/lib` directory of your application, if it is not already made visible to web applications by your servlet container.
  - **struts-config\_1\_1.dtd** - The document type descriptor (DTD) for the Struts configuration file (which is typically named `/WEB-INF/struts-config.xml`). Your configuration file will be validated against an internal copy of this DTD -- this copy is available for reference purposes only.
  - **web-app\_2\_2.dtd** - The document type descriptor (DTD) for web.xml files conforming to the Servlet 2.2 specification. This copy is for reference purposes only.
  - **web-app\_2\_3.dtd** - The document type descriptor (DTD) for web.xml files conforming to the Servlet 2.3 specification. This copy is for reference purposes only.
- **webapps/** - Web Application Archive (WAR) files for the web applications that are included with Struts.

## What's New?

### DEPRECATIONS:

- `struts-config.dtd` in favor of `struts-config_1_1.dtd`.

**COMMONS PACKAGES:** Several components of Struts 1.0 have been found to be useful in general Java development (and not just useful for building Struts-based web applications), and have been migrated into the [Jakarta Commons Project](#). As a result, the current development version of Struts has been modified to rely on the Commons packages containing these classes, rather than the Struts internal versions. In nearly every case, this involved changing only the import statements at the top of your classes. Any applications that utilize these classes will need to be modified in the same way. The following Commons packages contain the replacements for the corresponding Struts 1.0 classes:

- **BeanUtils Package** (`org.apache.commons.beanutils`) -  
`org.apache.struts.utils.BeanUtils`,  
`org.apache.struts.utils.ConvertUtils`, and  
`org.apache.struts.utils.PropertyUtils`.
- **Collections Package** (`org.apache.commons.collections`) -  
`org.apache.struts.util.ArrayStack`,  
`org.apache.struts.util.FastArrayList`,  
`org.apache.struts.util.FastHashMap`,  
`org.apache.struts.util.FastTreeMap`.
- **Digester Package** - (`org.apache.commons.digester`) -  
`org.apache.struts.digester.*`.

**XML PARSER PREREQUISITE UPDATED:** Struts now depends on an XML parser that conforms to the JAXP/1.1 (rather than JAXP/1.0) APIs. Parser known to work include the JAXP/1.1 reference implementation, and Xerces 1.3.1.

**CONTRIB Directory:** A new directory (`contrib`) in the CVS source repository has been added to accumulate Struts add-on extensions that are generally useful but have not yet been integrated into the standard code base.

- `ValidatorForm` - Client and Server-side validation library.
- `Tiles` - Advanced templating library (see `Struts-Tiles.war`).
- `Service Manager` - Add custom services without subclassing controller.

**UNIT TESTING SUPPORT:** Support for running unit tests on Struts components and custom tags is being added, utilizing the [Jakarta Cactus](#) product.

The following new features have been added to the basic controller framework (package `org.apache.struts.action`):

- The new `ActionMessages` class will support a superset of the capabilities of `ActionErrors`, and will be useful as a collection of general purpose messages, not just errors.

The following new features have been added to the utility classes (package `org.apache.struts.util`):

- `LocalStrings`: Correct message regarding replaceable parameter so that it does not append an extraneous character.

- Add LabelValueBean class. This defines a collection of name/value pairs that can be used with the <html:options> tag, and elsewhere.
- MessageResources: Escape any single quote characters that are included in the specified message string.
- Allow a transaction token to be the only parameter in computeParameters().
- Change RequestUtils to encode ampersands when building a query string.

The following new features have been added to the *struts-bean* custom tag library (package `org.apache.struts.taglib.bean`):

- Add format, locale and bundle attributes to bean:write to support values formatting according to current user locale, format string from attribute or format string from string resources.
- Correct the generated scripting variable type when the <bean:cookie>, <bean:header>, or <bean:parameter> tag is used with the "multiple" attribute.
- Added name, property, and scope attributes to the <bean:message> tag, so that the message source key can be obtained dynamically from a bean or bean property.

The following new features have been added to the *struts-html* custom tag library (package `org.apache.struts.taglib.html`):

- On the Options tag, if the property specified by the "property" attribute returns null, it now throws an error message that indicates what the real problem is, rather than causing an NPE.
- Added 'style' and 'styleClass' attributes for <html:option> and <html:options> tags.
- Added 'name', 'property' and 'scope' attributes to <bean:message> so that the message resource key can be obtained dynamically from a bean.
- Added a new <html:messages> tag to iterate through a message collection in the new ActionMessages class.
- ActionForm will now call reset ( ) if it instantiates the ActionForm bean.
- Added indexed property to the SubmitTag, SelectTag, LinkTag.java, CheckboxTag, ButtonTag, ImageTag, RadioTag, and TextArea.Tag.

The following new features have been added to the *struts-logic* custom tag library (package `org.apache.struts.taglib.logic`):

- Added <logic:empty> and <logic:notEmpty> tags, which are similar to <logic:present> and <logic:notPresent> except for the treatment of empty strings.

The following new features have been added to the *struts-template* custom tag library (package `org.apache.struts.taglib.template`):

- None.

The following new features have been added to the Struts Documentation application (and corresponding contents on the Struts web site):

- Move Tag Library documentation into User's Guide.
- Added Kickstart FAQ.
- Added Reference copy of 1.0 JavaDoc.
- Revised the example page in the User's Guide (Building View Components) to reflect current practice.
- Revised installation instructions for SilverStream and Resin.

## Changes and Bug Fixes

The following changes and bug fixes have occurred in the configuration files related to Struts:

- Remove deprecated support for the old (Struts 0.5) configuration file format.

The following changes and bug fixes have occurred in the basic controller framework (package `org.apache.struts.action`):

- Add `InvokeAction` and `CreateActionForm` methods to allow direct chaining of Actions.
- Add `ContextHelper` to expose framework elements to alternate presentation layers.
- `ActionForms` and related classes now use a `StringBuffer` when responding a `toString` request in order to conserve resources.
- Add standard `LookupDispatchAction` to help select between internationalized buttons.
- Modify `ActionForm` class to use `ActionServletWrapper` rather than expose `ActionServlet`.
- Add `ActionServletWrapper` class. Used by `ActionForm` to prevent the Public String properties of `ActionServlet` from being changed via a query string.
- Unconditionally pass the selected mapping as a request attribute under key `Action.MAPPING_KEY`, even if no form bean is specified.
- Avoid a `NullPointerException` in corner cases caused by failed initialization of `ActionServlet`.
- The `ActionForm` class is now truly serializable, because the two non-serializable instance variables (`servlet` and `multipartRequestHandler`) have been made transient. However, if you actually do serialize and deserialize such instances, it is your responsibility to reset these two properties.
- Removed deprecated Struts 0.5 methods, and support for the Struts 0.5 configuration file format.
- The initial order a property/key is added in is now maintained by `ActionMessages` class.

The following changes and bug fixes have occurred in the file upload package (package `org.apache.struts.upload`):

- Correct `MultiboxTagdoAfterBody()` to return `SKIP_BODY` instead of `SKIP_PAGE`.
- Fixed lost byte problem in `BufferedMultipartInputStream`
- Fixed `ArrayIndexOutOfBoundsException` situations
- Better reporting for premature closing of input streams while reading multipart requests.
- Additional fix for file corruption problem with uploads and new line characters.

The following changes and bug fixes have occurred in the utilities (package `org.apache.struts.util`):

- Modify `RequestUtils` class to use `ActionServletWrapper` rather than expose `ActionServlet`.
- Added error message for the `getActionErrors` and `getActionMessages` method.
- Added a `getActionErrors` and `getActionMessages` methods to generate the correct corresponding object based on the object retrieved from request scope based on the key passed in.
- The logic for creating an `ActionErrors` or `ActionMessages` object has been moved to a utility method in `RequestUtils`. The `JspException` message is also generated in `RequestUtils`.
- `ConvertUtils.convertCharacter()` will now detect empty strings and return the default value.

The following changes and bug fixes have occurred in the *struts-bean* custom tag library

(package org.apache.struts.taglib.bean):

- The `<html:errors>` when the property tag is specified, errors are no longer printed if the specified property has no errors. Previously errors were always printed ! Future enhancements would include additional attributes to always turn off the header or footer.
- Made the remaining helper methods "protected" rather than "private".

The following changes and bug fixes have occurred in the *struts-html* custom tag library (package org.apache.struts.taglib.html):

- Fixed FormTag to exclude query string when identifying action mapping name.
- Added the 'align' attribute to the `<html:image>` tag.
- Added indexed attribute to ImageTag, RadioTag, and TextAreaTag.
- Added MessagesTag.
- Correctly URLEncode the query string parameter value on ImgTag, even if there is only a single parameter.

The following changes and bug fixes have occurred in the *struts-logic* custom tag library (package org.apache.struts.taglib.logic):

- None.

The following changes and bug fixes to the Struts Documentation application (and corresponding contents on the Struts web site) have occurred:

- In the HTML tag documentation, expand to cover using indexed properties with iterate.
- Add installation notes for Jetty.
- In the Tag Developers Guide, add more detail regarding file upload requirements.
- In the Introduction, added references to basic background material.
- In Building View Components, clarify that additional i18n support may be provided by the browser, and is outside the scope of the framework.
- In Building Controller Components, document 'validating' init-param, add defaults for various parameters, clarify that some web.xml settings are not Struts-specific.
- Reorganized to separate 1.0 material from nightly build material.
- Expanded Resources section.
- Various updates regarding other release notes.

The following changes and bug fixes to the Struts Example Application have occurred:

- Add Russian and Japanese translations of the application resources and set the character set for the example JSP pages to "UTF-8" so that it can display either English or Japanese.
- Exchange "name" for "attribute" properties for Edit mappings in Struts configuration file.
- Remove references to saving database data from "tour" document, since this functionality was removed.

The following changes and bug fixes to the Struts Template Example Application have occurred:

- None.

The following changes and bug fixes to the Struts Exercise Taglib Example Application have occurred:



- Added test case for `<html:select>` using `<html:options>` based on a collection saved in the page context.

Next: [Installation](#)

---

*Copyright (c) 2000-2002, Apache Software Foundation*

Powered by  
**Struts**



## User Guide

[Table of Contents](#)

[Preface](#)

[Introduction](#)

[Model](#)

[Components](#)

[View](#)

[Components](#)

[Controller](#)

[Components](#)

[Configuration](#)

[Release Notes](#)

[Installation](#)

## Developer Guides

[Bean Tags](#)

[HTML Tags](#)

[Logic Tags](#)

[Nested Tags](#)

[Template Tags](#)

[Tiles Tags](#)

[Utilities](#)

[Validator](#)

## Quick Links

[Welcome](#)

[News and Status](#)

[Resources](#)

[User and  
Developer  
Guides \\*](#)

[FAQs and  
HowTos](#)

## Introduction

This document contains the release notes for **Version 1.0.2** of the Struts Framework, and covers changes that have taken place since [Version 1.0.1](#) was released. The following sections cover [New Features](#) and [Changes](#) to Struts.

This version covers bug-fixes only; no new functionality.

## What's Included?

The binary distribution of this release includes the following files relevant to Struts 1.0.2:

- **INSTALL** - Brief installation instructions. See the Struts Documentation Application, or online at <http://jakarta.apache.org/struts/> for more information.
- **LICENSE** - The Apache Software Foundation license that defines the terms under which you can use Struts (and other software licensed by Apache).
- **README** - A brief introduction to Struts.
- **lib/** - Directory containing files you will need in your own applications. The individual files of interest are:
  - **struts.jar** - JAR file that contains the compiled Java classes for version 1.0.2 of Struts. You must place this file in the `/WEB-INF/lib` directory of your web application.
  - **struts-xxxxx.tld** - The tag library descriptor files for the Struts 1.0.2 tag libraries (bean, html, logic, and template). You must place these files in the `/WEB-INF` directory of your web application, and reference them with appropriate `<taglib>` directives in your web.xml file.
  - **jdbc2\_0-stdext.jar** - The JDBC 2.0 Optional Package API classes (package `javax.sql`). You will need to include this file in the `/WEB-INF/lib` directory of your application, if it is not already made visible to web applications by your servlet container.
  - **struts-config\_1\_0.dtd** - The document type descriptor (DTD) for the Struts configuration file (which is typically named `/WEB-INF/struts-config.xml`). Your configuration file will be validated against an internal copy of this DTD -- this copy is available for reference purposes only.
  - **web-app\_2\_2.dtd** - The document type descriptor (DTD) for web.xml files conforming to the Servlet 2.2 specification. This copy is for reference purposes only.
  - **web-app\_2\_3.dtd** - The document type descriptor (DTD) for web.xml files conforming to the Servlet 2.3 specification. This copy is for reference purposes only.
- **webapps/** - Web Application Archive (WAR) files for the web applications that are included with Struts.

The struts 0.5 milestone API release is no longer supported, and will be removed as of Struts 1.1.

## What's New?

No new features have been added in this release.

## Changes and Bug Fixes

The following changes and bug fixes have occurred in the basic controller framework (package `org.apache.struts.action`):

- The `ActionForm.getMultipartRequestHandler()` method is now public rather than protected, to restore compatibility with the Tiles extension.

---

*Copyright (c) 2000-2002, Apache Software Foundation*

Powered by  
**Struts**



## User Guide

[Table of Contents](#)

[Preface](#)

[Introduction](#)

[Model](#)

[Components](#)

[View](#)

[Components](#)

[Controller](#)

[Components](#)

[Configuration](#)

[Release Notes](#)

[Installation](#)

## Developer Guides

[Bean Tags](#)

[HTML Tags](#)

[Logic Tags](#)

[Nested Tags](#)

[Template Tags](#)

[Tiles Tags](#)

[Utilities](#)

[Validator](#)

## Quick Links

[Welcome](#)

[News and Status](#)

[Resources](#)

[User and  
Developer  
Guides \\*](#)

[FAQs and  
HowTos](#)

## Introduction

This document contains the release notes for **Version 1.0.1** of the Struts Framework, and covers changes that have taken place since [Version 1.0](#) was released. The following sections cover [New Features](#) and [Changes](#) to Struts.

This version covers bug-fixes only; no new functionality.

## What's Included?

The binary distribution of this release includes the following files relevant to Struts 1.0.1:

- **INSTALL** - Brief installation instructions. See the Struts Documentation Application, or online at <http://jakarta.apache.org/struts/> for more information.
- **LICENSE** - The Apache Software Foundation license that defines the terms under which you can use Struts (and other software licensed by Apache).
- **README** - A brief introduction to Struts.
- **lib/** - Directory containing files you will need in your own applications. The individual files of interest are:
  - **struts.jar** - JAR file that contains the compiled Java classes for version 1.0 of Struts. You must place this file in the `/WEB-INF/lib` directory of your web application.
  - **struts-xxxxx.tld** - The tag library descriptor files for the Struts 1.0.1 tag libraries (bean, html, logic, and template). You must place these files in the `/WEB-INF` directory of your web application, and reference them with appropriate `<taglib>` directives in your web.xml file.
  - **jdbc2\_0-stdext.jar** - The JDBC 2.0 Optional Package API classes (package `javax.sql`). You will need to include this file in the `/WEB-INF/lib` directory of your application, if it is not already made visible to web applications by your servlet container.
  - **struts-config\_1\_0.dtd** - The document type descriptor (DTD) for the Struts configuration file (which is typically named `/WEB-INF/struts-config.xml`). Your configuration file will be validated against an internal copy of this DTD -- this copy is available for reference purposes only.
  - **web-app\_2\_2.dtd** - The document type descriptor (DTD) for web.xml files conforming to the Servlet 2.2 specification. This copy is for reference purposes only.
  - **web-app\_2\_3.dtd** - The document type descriptor (DTD) for web.xml files conforming to the Servlet 2.3 specification. This copy is for reference purposes only.
- **webapps/** - Web Application Archive (WAR) files for the web applications that are included with Struts.

The struts 0.5 milestone API release is no longer supported, and has been removed as of Struts 1.1.

## What's New?

The following new features have been added to the basic controller framework (package `org.apache.struts.action`):

The following new features have been added to the utility classes (package `org.apache.struts.util`):

- `LocalStrings`: Correct message regarding replaceable parameter so that it does not append an extraneous character.
- `MessageResources`: Escape any single quote characters that are included in the specified message string.
- Allow a transaction token to be the only parameter in `computeParameters()`.
- Change `RequestUtils` to encode ampersands when building a query string.

The following new features have been added to the *struts-html* custom tag library (package `org.apache.struts.taglib.html`):

The following new features have been added to the *struts-logic* custom tag library (package `org.apache.struts.taglib.logic`):

The following new features have been added to the *struts-template* custom tag library (package `org.apache.struts.taglib.template`):

The following new features have been added to the Struts Documentation application (and corresponding contents on the Struts web site):

## Changes and Bug Fixes

The following changes and bug fixes have occurred in the configuration files related to Struts:

The following changes and bug fixes have occurred in the basic controller framework (package `org.apache.struts.action`):

- Modify `ActionForm` class to use `ActionServletWrapper` rather than expose `ActionServlet`.
- Add `ActionServletWrapper` class. Used by `ActionForm` to prevent the Public String properties of `ActionServlet` from being changed via a query string.
- Unconditionally pass the selected mapping as a request attribute under key `Action.MAPPING_KEY`, even if no form bean is specified.
- Avoid a `NullPointerException` in corner cases caused by failed initialization of `ActionServlet`.

The following changes and bug fixes have occurred in the file upload package (package `org.apache.struts.upload`):

- Fixed lost byte problem in `BufferedMultipartInputStream`
- Fixed `ArrayIndexOutOfBoundsException` situations
- Better reporting for premature closing of input streams while reading multipart requests.
- Additional fix for file corruption problem with uploads and new line characters.

The following changes and bug fixes have occurred in the utilities (package `org.apache.struts.util`):

- Modify RequestUtils class to use ActionServletWrapper rather than expose ActionServlet.
- ConvertUtils.convertCharacter( ) will now detect empty strings and return the default value.

The following changes and bug fixes have occurred in the *struts-bean* custom tag library (package `org.apache.struts.taglib.bean`):

- Correct the generated scripting variable type when the `<bean:cookie>`, `<bean:header>`, or `<bean:parameter>` tag is used with the "multiple" attribute.

The following changes and bug fixes have occurred in the *struts-html* custom tag library (package `org.apache.struts.taglib.html`):

- Fixed FormTag to exclude query string when identifying action mapping name.
- Correct MultiboxTag.doAfterBody() to return SKIP\_BODY instead of SKIP\_PAGE.
- Added the 'align' attribute to the `<html:image>` tag.
- On the Options tag, if the property specified by the "property" attribute returns null, it now throws an error message that indicates what the real problem is, rather than causing an NPE.
- Added 'style' and 'styleClass' attributes for `<html:option>` and `<html:options>` tags.
- Correctly URLEncode the query string parameter value on ImgTag, even if there is only a single parameter.

The following changes and bug fixes have occurred in the *struts-logic* custom tag library (package `org.apache.struts.taglib.logic`):

- The `<html:errors>` when the property tag is specified, errors are no longer printed if the specified property has no errors. Previously errors were always printed! Future enhancements would include additional attributes to always turn off the header or footer.

The following changes and bug fixes to the Struts Documentation application (and corresponding contents on the Struts web site) have occurred:

- Add installation notes for Jetty.
- In the Tag Developers Guide, add more detail regarding file upload requirements.
- In the Introduction, added references to basic background material.
- In Building View Components, clarify that additional i18n support may be provided by the browser, and is outside the scope of the framework.
- In Building Controller Components, document 'validating' init-param, add defaults for various parameters, clarify that some web.xml settings are not Struts-specific.
- Correct the example page in the User's Guide (Building View Components) to reflect current practice.
- Revised installation instructions for SilverStream and Resin.

The following changes and bug fixes to the Struts Example Application have occurred:

- Remove references to saving database data from "tour" document, since this functionality was removed.

The following changes and bug fixes to the Struts Template Example Application have occurred:

The following changes and bug fixes to the Struts Exercise Taglib Example Application have

occurred:

- Added test case for `<html:select>` using `<html:options>` based on a collection saved in the page context.

---

*Copyright (c) 2000-2002, Apache Software Foundation*

Powered by  
**Struts**



## User Guide

[Table of Contents](#)

[Preface](#)

[Introduction](#)

[Model Components](#)

[View Components](#)

[Controller Components](#)

[Configuration](#)

[Release Notes](#)

[Installation](#)

## Developer Guides

[Bean Tags](#)

[HTML Tags](#)

[Logic Tags](#)

[Nested Tags](#)

[Template Tags](#)

[Tiles Tags](#)

[Utilities](#)

[Validator](#)

## Quick Links

[Welcome](#)

[News and Status](#)

[Resources](#)

[User and Developer Guides \\*](#)

[FAQs and HowTos](#)

## Introduction

This document contains the release notes for **Version 1.0** of the Struts Framework, and covers changes that have taken place since [Version 1.0-beta-3](#) was released. The following sections cover [New Features](#) and [Changes](#) to Struts.

## What's Included?

The binary distribution of this release includes the following files relevant to Struts 1.0:

- **INSTALL** - Brief installation instructions. See the Struts Documentation Application, or online at <http://jakarta.apache.org/struts/> for more information.
- **LICENSE** - The Apache Software Foundation license that defines the terms under which you can use Struts (and other software licensed by Apache).
- **README** - A brief introduction to Struts.
- **lib/** - Directory containing files you will need in your own applications. The individual files of interest are:
  - **struts.jar** - JAR file that contains the compiled Java classes for both version 0.5 and 1.0 of Struts. You must place this file in the `/WEB-INF/lib` directory of your web application.
  - **struts-xxxxx.tld** - The tag library descriptor files for the Struts 1.0 tag libraries (bean, html, logic, and template). You must place these files in the `/WEB-INF` directory of your web application, and reference them with appropriate `<taglib>` directives in your web.xml file. **NOTE** - The struts-form.tld file is deprecated; you should use the struts-html.tld file instead.
  - **jdbc2\_0-stdext.jar** - The JDBC 2.0 Optional Package API classes (package `javax.sql`). You will need to include this file in the `/WEB-INF/lib` directory of your application, if it is not already made visible to web applications by your servlet container.
  - **struts-config\_1\_0.dtd** - The document type descriptor (DTD) for the Struts configuration file (which is typically named `/WEB-INF/struts-config.xml`). Your configuration file will be validated against an internal copy of this DTD -- this copy is available for reference purposes only.
  - **web-app\_2\_2.dtd** - The document type descriptor (DTD) for web.xml files conforming to the Servlet 2.2 specification. This copy is for reference purposes only.
  - **web-app\_2\_3.dtd** - The document type descriptor (DTD) for web.xml files conforming to the Servlet 2.3 specification. This copy is for reference purposes only.
- **webapps/** - Web Application Archive (WAR) files for the web applications that are included with Struts.

For backwards compatibility only, the binary distribution also includes the following



files that conform to the Struts 0.5 milestone release APIs. Usage of these files and APIs is deprecated, and they will be removed from releases after Struts 1.0:

- `lib/` - Directory containing files you will need in your own applications. The individual files of interest are:
  - `struts.jar` - JAR file that contains the compiled Java classes for both version 0.5 and 1.0 of Struts. You must place this file in the `/WEB-INF/lib` directory of your web application.
  - `struts.tld` - The tag library descriptor file for the 0.5 version of the Struts tags. You must place this file in the `/WEB-INF` directory of your web application, and reference it with appropriate `<taglib>` directives in your `web.xml` file.

## What's New?

**DEPRECATIONS** - The entire custom tag library that is documented in `struts.tld` has been deprecated. These tags correspond to the Struts 0.5 functionality that is also deprecated, and have been replaced (and considerably enhanced) in the various individual tag libraries.

**DEPRECATIONS** - The entire custom tag library that is documented in `struts-form.tld` has been deprecated because this library has been renamed `struts-html.tld` instead.

**DEPRECATIONS** - Several classes in the `org.apache.struts.util` package have been marked as deprecated in their entirety, because they will be replaced by versions from the Jakarta Commons project once those packages are released. These deprecated classes will **not** be removed until a release after Struts 1.1. In general, the only change required inside user code using these classes will be to update the `import` statement. The following classes are involved:

- `ArrayStack`
- `BeanUtils`
- `ConvertUtils`
- `FastArrayList`
- `FastHashMap`
- `FastTreeMap`
- `GenericConnection`
- `GenericDataSource`
- `PropertyUtils`

The following new features have been added to the basic controller framework (package `org.apache.struts.action`):

The following new features have been added to the utility classes (package `org.apache.struts.util`):

The following new features have been added to the *struts-html* custom tag library (package `org.apache.struts.taglib.html`):

- The `<html:link>` tag now supports the optional `title` attribute, that causes some browsers to display alternate text when the mouse is hovered over a hyperlink.

The following new features have been added to the *struts-logic* custom tag library (package `org.apache.struts.taglib.logic`):

- The `<logic:iterate>` tag now supports a new attribute, `indexId`. This attribute names a page scope attribute (and corresponding scripting variable) that will be exposed in the nested body of the iteration, which will contain the current loop index as a `java.lang.Integer`.

The following new features have been added to the *struts-template* custom tag library (package `org.apache.struts.taglib.template`):

The following new features have been added to the Struts Documentation application (and corresponding contents on the Struts web site):

## Changes and Bug Fixes

The following changes and bug fixes have occurred in the configuration files related to Struts:

- The documentation about configuring a data source element in the `struts-config.xml` file has been updated to emphasize the use of `<set-property>` elements nested inside the `<data-source>` element. No further additional attributes will be added to the `<data-source>` element in the DTD to support custom properties of particular data source implementations.

The following changes and bug fixes have occurred in the basic controller framework (package `org.apache.struts.action`):

- Several issues introduced with the internal wrapping of multipart requests have been fixed.
- The instance variables in the `ActionForm` base class are now transient, so that `ActionForm` instances can actually be serialized. **WARNING** - If you deserialize such an `ActionForm` instance and attempt to use it within the Struts framework, be sure to call `setServlet()` (and `setMultipartRequestHandler()` if appropriate) first.

The following changes and bug fixes have occurred in the file upload package (package `org.apache.struts.upload`):

- Fixed a bug that could cause corruption in the uploaded file, by converting sequences of `\r\n\n` into `\r\n\r\n`.

The following changes and bug fixes have occurred in the utilities (package `org.apache.struts.util`):

- Corrected a cut-and-paste typo that caused `NullPointerException` on attempts to use the new `pingQuery` property that was added in Struts 1.0-b3.
- Fix operation ordering so that `RequestUtils.lookup()` will throw a `JspException` when the specified bean is missing, in accordance with its JavaDoc documentation.
- `Digester` calls to `Rule.body()` methods will now trim leading and trailing whitespace first, consistent with the behavior of other body processing performed within the rules.

- Correct the calculation of an absolute URL from a context-relative path in `RequestUtils.absoluteURL()`.

The following changes and bug fixes have occurred in the *struts-bean* custom tag library (package `org.apache.struts.taglib.bean`):

- The `<bean:include>` tag will now pass a session identifier on to the included request (assuming that it is part of the same web application), even if the current request is maintaining session identity with cookies. Previously, this only worked if you were using URL rewriting.

The following changes and bug fixes have occurred in the *struts-html* custom tag library (package `org.apache.struts.taglib.html`):

- For all tags where Struts generates attributes that are allowed to be minimized in the HTML specification (i.e. checked, disabled, multiple, readonly, and selected), generate attribute values equal to the attribute name, as required by HTML 4.01, section 3.3.4. For example, generate `checked="checked"` instead of `checked="true"` for the "checked" attribute.
- Correctly generate a `<a name="my name"><>` element when the `linkName` attribute is used on the `<html:link>` tag.

The following changes and bug fixes have occurred in the *struts-logic* custom tag library (package `org.apache.struts.taglib.logic`):

- Update the processing performed by the `<logic:present>` and `<logic:notPresent>` tags to reflect the changed behavior of `RequestUtils.lookup()`.

The following changes and bug fixes to the Struts Documentation application (and corresponding contents on the Struts web site) have occurred:

- Links to subscribe and unsubscribe from the STRUTS-DEV and STRUTS-USER mailing lists have been added to the home page.
- Miscellaneous corrections to typos and hyperlinks.
- Cleaned up problems in the stylesheets used to create documentation pages that formerly emitted `<project>` tags in the generated HTML, and generated incorrect references to link colors in the navigation bar.
- Added a "Who We Are" page to the documentation.

The following changes and bug fixes to the Struts Example Application have occurred:

- The `name` and `page` attributes of the `<app:checkLogon>` tag now accept runtime expressions.

The following changes and bug fixes to the Struts Template Example Application have occurred:



## User Guide

[Table of Contents](#)

[Preface](#)

[Introduction](#)

[Model Components](#)

[View Components](#)

[Controller Components](#)

[Configuration](#)

[Release Notes](#)

[Installation](#)

## Developer Guides

[Bean Tags](#)

[HTML Tags](#)

[Logic Tags](#)

[Nested Tags](#)

[Template Tags](#)

[Tiles Tags](#)

[Utilities](#)

[Validator](#)

## Quick Links

[Welcome](#)

[News and Status](#)

[Resources](#)

[User and Developer Guides \\*](#)

[FAQs and HowTos](#)

## Introduction

This document contains the release notes for **Version 1.0-beta-3** of the Struts Framework, and covers changes that have taken place since [Version 1.0-beta-2](#) was released. The following sections cover [New Features](#) and [Changes](#) to Struts.

## What's Included?

The binary distribution of this release includes the following files relevant to Struts 1.0:

- **INSTALL** - Brief installation instructions. See the Struts Documentation Application, or online at <http://jakarta.apache.org/struts/> for more information.
- **LICENSE** - The Apache Software Foundation license that defines the terms under which you can use Struts (and other software licensed by Apache).
- **README** - A brief introduction to Struts.
- **lib/** - Directory containing files you will need in your own applications. The individual files of interest are:
  - **struts.jar** - JAR file that contains the compiled Java classes for both version 0.5 and 1.0 of Struts. You must place this file in the `/WEB-INF/lib` directory of your web application.
  - **struts-xxxxx.tld** - The tag library descriptor files for the Struts 1.0 tag libraries (bean, html, logic, and template). You must place these files in the `/WEB-INF` directory of your web application, and reference them with appropriate `<taglib>` directives in your web.xml file. **NOTE** - The struts-form.tld file is deprecated; you should use the struts-html.tld file instead.
  - **jdbc2\_0-stdext.jar** - The JDBC 2.0 Optional Package API classes (package `javax.sql`). You will need to include this file in the `/WEB-INF/lib` directory of your application, if it is not already made visible to web applications by your servlet container.
  - **struts-config\_1\_0.dtd** - The document type descriptor (DTD) for the Struts configuration file (which is typically named `/WEB-INF/struts-config.xml`). Your configuration file will be validated against an internal copy of this DTD -- this copy is available for reference purposes only.
  - **web-app\_2\_2.dtd** - The document type descriptor (DTD) for web.xml files conforming to the Servlet 2.2 specification. This copy is for reference purposes only.
  - **web-app\_2\_3.dtd** - The document type descriptor (DTD) for web.xml files conforming to the Servlet 2.3 specification. This copy is for reference purposes only.
- **webapps/** - Web Application Archive (WAR) files for the web applications that are included with Struts.

For backwards compatibility only, the binary distribution also includes the following

files that conform to the Struts 0.5 milestone release APIs. Usage of these files and APIs is deprecated, and they will be removed from releases after Struts 1.0:

- `lib/` - Directory containing files you will need in your own applications. The individual files of interest are:
  - `struts.jar` - JAR file that contains the compiled Java classes for both version 0.5 and 1.0 of Struts. You must place this file in the `/WEB-INF/lib` directory of your web application.
  - `struts.tld` - The tag library descriptor file for the 0.5 version of the Struts tags. You must place this file in the `/WEB-INF` directory of your web application, and reference it with appropriate `<taglib>` directives in your `web.xml` file.

## What's New?

**DEPRECATIONS** - The entire custom tag library that is documented in `struts.tld` has been deprecated. These tags correspond to the Struts 0.5 functionality that is also deprecated, and have been replaced (and considerably enhanced) in the various individual tag libraries.

**DEPRECATIONS** - The entire custom tag library that is documented in `struts-form.tld` has been deprecated because this library has been renamed `struts-html.tld` instead.

**DEPRECATIONS** - Several classes in the `org.apache.struts.util` package have been marked as deprecated in their entirety, because they will be replaced by versions from the Jakarta Commons project once those packages are released. These deprecated classes will **not** be removed until a release after Struts 1.1. In general, the only change required inside user code using these classes will be to update the `import` statement. The following classes are involved:

- `ArrayStack`
- `BeanUtils`
- `ConvertUtils`
- `FastArrayList`
- `FastHashMap`
- `FastTreeMap`
- `GenericConnection`
- `GenericDataSource`
- `PropertyUtils`

The following new features have been added to the basic controller framework (package `org.apache.struts.action`):

The following new features have been added to the utility classes (package `org.apache.struts.util`):

- `GenericDataSource` can now be configured with a "ping" type command (via the `pingCommand` and `pingQuery` properties) that will be executed before returning a connection from `getConnection()`. This can be used to detect stale connections due to timeouts or a database server restart. If the ping command fails, the corresponding connection will be thrown away, and a new one allocated.

The following new features have been added to the *struts-html* custom tag library (package `org.apache.struts.taglib.html`):

- It is now possible to use any object in the values and labels collections for the `<html:options>` tag.
- Added the missing `tabindex` attribute to the `<html:radio>` tag.
- On the `<html:img>`, `<html:link>`, and `<html:rewrite>` tags, you can now specify arbitrary object values in the Map used to include dynamic request attributes, as long as the `toString()` method renders the values appropriately.
- In all cases where "minimized" attributes were being generated (`checked`, `disabled`, `multiple`, `readonly`, and `selected`), the generated attribute has a value (such as `selected="true"`) for XML syntax compatibility.

The following new features have been added to the *struts-logic* custom tag library (package `org.apache.struts.taglib.logic`):

The following new features have been added to the *struts-template* custom tag library (package `org.apache.struts.taglib.template`):

The following new features have been added to the Struts Documentation application (and corresponding contents on the Struts web site):

## Changes and Bug Fixes

The following changes and bug fixes have occurred in the configuration files related to Struts:

- The DTD for `struts-config.xml` files had an incorrect ENTITY declaration for the `Location` element.
- The DTD for version 2.3 web application deployment descriptors has been updated to the most recent (Proposed Final Draft 2) version.

The following changes and bug fixes have occurred in the basic controller framework (package `org.apache.struts.action`):

The following changes and bug fixes have occurred in the utilities (package `org.apache.struts.util`):

- The `FastArrayMap()`, `FastHashMap()`, and `FastTreeMap()` classes not correctly implement the `clone()`, `equals()`, and `hashCode()` methods consistent with the requirements of the Java standard Collections APIs.
- `PropertyUtils` can now access public methods defined in nested interfaces.
- A misleading error message returned by `BeanUtils` has been corrected.
- Work around a problem compiling the `FastXXXXX` classes with the VAJ compiler, because the superclass already includes a private class named `Iterator`.
- Remove a JDK 1.3 dependency that prevented compiling `BeanUtils` under JDK 1.2.
- Generate `"&";` instead of `"&"` in query strings that contain more than one name/value pair.

The following changes and bug fixes have occurred in the *struts-html* custom tag library (package `org.apache.struts.taglib.html`):

- Restore the ability of the `<html:button>` tag to retrieve the button text from the nested body content (so that it can be easily internationalized).
- If the property name used in the `labelProperty` attribute of an `<html:options>` tag is invalid, report the correct property name in the error message.

The following changes and bug fixes have occurred in the *struts-logic* custom tag library (package `org.apache.struts.taglib.logic`):

The following changes and bug fixes to the Struts Documentation application (and corresponding contents on the Struts web site) have occurred:

- Fix a reference to an old Struts 0.5 tag in one of the User's Guide examples.
- Fix the stylesheet used to transform XML into HTML documentation so that nested `<body>` tags are not created.
- The DTD for Struts configuration files has been refined to highlight the fact that you should use `<set-property>` elements to configure your data source implementation.

The following changes and bug fixes to the Struts Example Application have occurred:

The following changes and bug fixes to the Struts Template Example Application have occurred:

---

*Copyright (c) 2000-2002, Apache Software Foundation*

Powered by  
**Struts**





## User Guide

[Table of Contents](#)

[Preface](#)

[Introduction](#)

[Model](#)

[Components](#)

[View](#)

[Components](#)

[Controller](#)

[Components](#)

[Configuration](#)

[Release Notes](#)

[Installation](#)

## Developer Guides

[Bean Tags](#)

[HTML Tags](#)

[Logic Tags](#)

[Nested Tags](#)

[Template Tags](#)

[Tiles Tags](#)

[Utilities](#)

[Validator](#)

## Quick Links

[Welcome](#)

[News and Status](#)

[Resources](#)

[User and  
Developer  
Guides \\*](#)

[FAQs and  
HowTos](#)

## Introduction

This document contains the release notes for **Version 1.0-beta-2** of the Struts Framework, and covers changes that have taken place since [Version 1.0-beta-1](#) was released. The following sections cover [New Features](#) and [Changes](#) to Struts.

## What's Included?

The binary distribution of this release includes the following files relevant to Struts 1.0:

- **INSTALL** - Brief installation instructions. See the Struts Documentation Application, or online at <http://jakarta.apache.org/struts/> for more information.
- **LICENSE** - The Apache Software Foundation license that defines the terms under which you can use Struts (and other software licensed by Apache).
- **README** - A brief introduction to Struts.
- **lib/** - Directory containing files you will need in your own applications. The individual files of interest are:
  - **struts.jar** - JAR file that contains the compiled Java classes for both version 0.5 and 1.0 of Struts. You must place this file in the `/WEB-INF/lib` directory of your web application.
  - **struts-xxxxx.tld** - The tag library descriptor files for the Struts 1.0 tag libraries (bean, html, logic, and template). You must place these files in the `/WEB-INF` directory of your web application, and reference them with appropriate `<taglib>` directives in your web.xml file. **NOTE** - The `struts-form.tld` file is deprecated; you should use the `struts-html.tld` file instead.
  - **jdbc2\_0-stdext.jar** - The JDBC 2.0 Optional Package API classes (package `javax.sql`). You will need to include this file in the `/WEB-INF/lib` directory of your application, if it is not already made visible to web applications by your servlet container.
  - **struts-config\_1\_0.dtd** - The document type descriptor (DTD) for the Struts configuration file (which is typically named `/WEB-INF/struts-config.xml`). Your configuration file will be validated against an internal copy of this DTD -- this copy is available for reference purposes only.
  - **web-app\_2\_2.dtd** - The document type descriptor (DTD) for web.xml files conforming to the Servlet 2.2 specification. This copy is for reference purposes only.
  - **web-app\_2\_3.dtd** - The document type descriptor (DTD) for web.xml files conforming to the Servlet 2.3 specification. This copy is for reference purposes only.
- **webapps/** - Web Application Archive (WAR) files for the web applications that are included with Struts.

For backwards compatibility only, the binary distribution also includes the following files that conform to the Struts 0.5 milestone release APIs. Usage of these files and APIs is deprecated, and they will be removed from releases after Struts 1.0:



- `lib/` - Directory containing files you will need in your own applications. The individual files of interest are:
  - `struts.jar` - JAR file that contains the compiled Java classes for both version 0.5 and 1.0 of Struts. You must place this file in the `/WEB-INF/lib` directory of your web application.
  - `struts.tld` - The tag library descriptor file for the 0.5 version of the Struts tags. You must place this file in the `/WEB-INF` directory of your web application, and reference it with appropriate `<taglib>` directives in your `web.xml` file.

## What's New?

**DEPRECATIONS** - The entire custom tag library that is documented in `struts.tld` has been deprecated. These tags correspond to the Struts 0.5 functionality that is also deprecated, and have been replaced (and considerably enhanced) in the various individual tag libraries.

The build procedure for compiling the Struts source distribution has been revised, and now depends on having **Ant 1.2** (or later) installed, with the `$ANT_HOME/bin` directory on your path. Further information can be found in the [Installation](#) documentation.

The following new features have been added to the basic controller framework (package `org.apache.struts.action`):

- You can now specify that an `<action>` element should invoke an existing servlet or JSP page resource, rather than calling an Action class, by using the `include` attribute rather than the `type` attribute. The standard form bean processing provided by the controller is still performed first, if you have configured it, so the included resource can benefit from this processing if it wishes to.
- The `initDataSources()` method can now throw a `ServletException` to report that an initialization error has occurred. Previously, such errors were logged but otherwise ignored.
- It is now possible to integrate business logic that is already encapsulated as a servlet or JSP page, via use of two new standard actions:  
`org.apache.struts.actions.ForwardAction` and  
`org.apache.struts.actions.IncludeAction`. These actions let you take advantage of the standard processing performed by the controller servlet (including form bean population and calling the `validate()` method), but not have to write Java code to perform (or wrap) the required business logic.
- A wrapper class has been added around the standard `HttpServletRequest` for handling multipart requests as identically as possible to standard requests, including processing request parameters, populating form beans, transaction tokens, and checking for cancellations.

The following new features have been added to the utility classes (package `org.apache.struts.util`):

- `PropertyUtils` can now correctly locate public methods defined in a nested interface that is implemented by a bean.
- `PropertyUtils` methods now throw `IllegalArgumentException` when you pass a null bean reference or property name.
- **DEPRECATIONS** - The following classes have been deprecated in their entirety, because they will be replaced by corresponding classes (with identical functionality) from the [Jakarta Commons Project](#) at some point after Struts 1.0 final release. The only change that will ultimately be required in user code is to change the package names on

the import clauses:

- `org.apache.struts.util.FastArrayList`
- `org.apache.struts.util.FastHashMap`
- `org.apache.struts.util.FastTreeMap`
- `org.apache.struts.util.BeanUtils`
- `org.apache.struts.util.ConvertUtils`
- `org.apache.struts.util.PropertyUtils`

The following new features have been added to the *struts-html* custom tag library (package `org.apache.struts.taglib.html`):

- The `<html:link>` tag now accepts an optional `anchor` attribute, to allow the inclusion of a (possibly calculated) anchor ("`#xxx`") in the generated hyperlink.
- The `<html:base>` tag now accepts an optional `target` attribute.
- The `<html:image>` tag now accepts an optional `border` attribute, to define the border with around this image.
- You can now request that the `<html:link>` tag include any current transaction control token in the generated hyperlink, by specifying the `transaction` attribute with a value of `true`.
- The `<html:options>` tag now supports Enumeration for the `collection` property.
- The `<html:form>` tag now creates attributes for the tag itself, and the form bean, in request scope instead of page scope. Among other benefits, this allows you to nest the fields of a form inside a separate page that is accessed via a template or a `<jsp:include>` tag.
- The `styleId` attribute has been added to all of the tags where the corresponding `id` tag is relevant, to identify a specific tag for the purposes of stylesheet references.
- The `<html:file>` tag now supports the `size` attribute to set the size of the file list field.

The following new features have been added to the *struts-logic* custom tag library (package `org.apache.struts.taglib.logic`):

- The `<logic:iterate>` tag now supports Enumeration for the `collection` property.
- The `<logic:iterate>` tag now exposes the current iteration index to nested tags, through a call to the `getIndex()` method.

The following new features have been added to the *struts-template* custom tag library (package `org.apache.struts.taglib.template`):

- Attribute getter methods have been added to all of the custom tag implementation classes to facilitate reuse.
- The `<template:get>` tag now has an optional "flush" attribute that causes the response to be committed prior to performing the include, if set to "true". This allows working around problems on broken servlet containers.

The following new features have been added to the Struts Documentation application (and corresponding contents on the Struts web site):

- Platform specific installation notes for a wide variety of application server and servlet container environments have been accumulated and published.

## Changes and Bug Fixes

The following changes and bug fixes have occurred in the basic controller framework (package `org.apache.struts.action`):

- `DiskMultipartRequestHandler` now tries to retrieve the temporary directory provided by the servlet container before all other possible temporary directories.

The following changes and bug fixes have occurred in the utilities (package `org.apache.struts.util`):

- The `computeURL()` method now returns a `MalformedURLException` if a URL cannot be created. Previously, this case returned `null` with no error message, making some problems difficult to track down.

The following changes and bug fixes have occurred in the *struts-html* custom tag library (package `org.apache.struts.taglib.html`):

- The `<html:image>` tag now correctly generates a closing double quote on the name attribute.
- The `<html:form>` tag now generates a hyperlink that includes any query parameters that were included in the original hyperlink value.
- The `<html:link>` tag now correctly places any specified anchor ("`#xxx`") in the generated hyperlink.
- The JavaScript code generated to implement the `focus` attribute of the `<html:form>` tag now works even when you use nested or indexed property expressions. Previously, this would generate an invalid JavaScript reference to the field.
- The `<html:checkbox>` tag now conforms to its documentation, and sends the specified value to the server if this checkbox is checked at submit time. In addition, a default value of `on` is sent if no value attribute is specified.
- The hyperlinks created by the `<html:link>` and `<html:redirect>` tags now properly omit the port number if it is the default port for the current request scheme (80 for `http`, or 443 for `https`). Among other things, this corrects session management behavior on the standard port numbers.
- The `focus` attribute of the `<html:form>` tag now works when the corresponding input field is a radio button, or is otherwise indexed.
- The `disabled` and `readonly` attributes have been added to all HTML-rendering tags where they are relevant.

The following changes and bug fixes have occurred in the *struts-logic* custom tag library (package `org.apache.struts.taglib.logic`):

- Hyperlink processing in the `<logic:forward>` tag is now identical to that performed by the Struts Controller Servlet when it processes an `ActionForward` that is returned by an `Action`.

The following changes and bug fixes to the Struts Documentation application (and corresponding contents on the Struts web site) have occurred:

- The Java types of collections (and arrays) over which the `<logic:iterate>` tag can run are now documented.
- The `<bean:define>` documentation now mentions the JSP 1.1 Specification restriction on using more than one `id` attribute with the same value in the same page.

The following changes and bug fixes to the Struts Example Application have occurred:

- The Java source code of these applications is now included in the corresponding WAR

files, in subdirectory WEB-INF/src.

- Excessive filter() calls in LinkSubscriptionTag have been eliminated.
- Calls to the deprecated BeanUtils.filter( ) have been replaced by calls to ResponseUtils.filter( ).
- Removed any attempt to save the pseudo-database at application shutdown, because there is no portable mechanism to accomplish this task.

The following changes and bug fixes to the Struts Template Example Application have occurred:

- Refactored the example pages to eliminate the creation of redundant <html>, <head>, and <body> tags.
- If <template:get> or <template:insert> throws an exception, do not overwrite any "real" exception that has already been saved.
- The text of the various pages in the example has been updated so that they are not identical.

---

*Copyright (c) 2000-2002, Apache Software Foundation*

**Powered by  
Struts**



## User Guide

[Table of Contents](#)

[Preface](#)

[Introduction](#)

[Model](#)

[Components](#)

[View](#)

[Components](#)

[Controller](#)

[Components](#)

[Configuration](#)

[Release Notes](#)

[Installation](#)

## Developer Guides

[Bean Tags](#)

[HTML Tags](#)

[Logic Tags](#)

[Nested Tags](#)

[Template Tags](#)

[Tiles Tags](#)

[Utilities](#)

[Validator](#)

## Quick Links

[Welcome](#)

[News and Status](#)

[Resources](#)

[User and  
Developer  
Guides \\*](#)

[FAQs and  
HowTos](#)

## Introduction

This document contains the release notes for **Version 1.0-beta-1** of the Struts Framework. The following sections cover [New Features](#) and [Changes](#) since the **Version 0.5** of Struts was made available.

One global new feature to take note of is that Struts 1.0 requires a Java2 (JDK 1.2 or later) platform on which to run.

## What's New?

The following major new features have been added to the controller servlet and associated classes (package `org.apache.struts.action`):

- A new configuration file format, including the [Document Type Definition \(DTD\)](#) it is based on, is available for configuring the Struts controller servlet. Support for the previous format is still present, but will be phased out by the 1.0 release.
- If enabled by setting the `locale` servlet initialization parameter to `true`, the controller servlet will now calculate a Locale for this user (based on the Locale returned by the servlet container, or by the HTTP headers included with the request) and store it in the user's session, unless the application has stored one there already.
- Application `Action` instances now have lifecycle support. The controller servlet will call `setServlet()` with a non-null argument when the instance is first created, and call it with a null argument when the instance is being shut down.
- The collection of "standard" Struts actions (in package `org.apache.struts.actions`) is kicked off with a set of simple actions that allow on-the-fly changes in the form bean, forward, and mapping definitions registered with the controller servlet. These actions would normally be configured behind security constraints to prevent interference with the operation of your application but can be very useful during development.
- A new representation of error messages (`ActionError` and `ActionErrors`) has been created that allows errors to be associated with individual fields, and stores parameter replacement values along with the messages keys.
- The `ActionForm` interface has been converted to a base class, with default implementations of some useful functionality. In addition, the new class provides two `validate()` methods that take the current mapping and current request as arguments, in order to provide access to more context information than just the properties of this particular form bean.
- The new `ActionForm` class also provides two `reset()` methods that take the current mapping and current request as arguments, in order to provide access to more context information (particularly important on multi-page forms so that the form bean knows which properties to reset). Among other things, use of the `reset()` methods can prevent problems with checkbox fields -- simply reset boolean fields to `false` and they will be set to `true` (during auto-population of the form bean properties) only if the checkbox was actually checked.
- All servlet context attributes created by the Struts controller servlet itself (but not the Struts Example application) now implement `java.io.Serializable`.

- The `Action` class now includes support for "transactional tokens", so that you can detect cases where the user submitted a form, then went back and resubmitted the form again.

The following major new features have been added to the *struts-bean* custom tag library (package `org.apache.struts.taglib.bean`):

- A new custom tag, `<bean:page>`, is available to expose key items from the page context associated with the current page as scripting variables, and as page-scope beans. For example, you can use the following sequence to render the server information string returned by our servlet context:

```
<bean:page id="app" property="application"/>
<bean:write name="app" property="serverInfo"/>
```

- A new custom tag, `<bean:struts>`, is available to expose internal Struts configuration objects (form bean, forward, and mapping definitions) as scripting variables and page-scope beans. For example, you can use the following sequence to render the actual context-relative path of an `ActionForward` object:

```
<bean:struts id="link" forward="success"/>
<bean:write name="link" property="path"/>
```

- All of the tags in this library that accept a name attribute referring to a JSP bean now also accept an optional scope attribute to define the scope in which to search for that bean. If not specified, all scopes are searched.
- The `<bean:size>` tag will create a bean that stores the number of elements of an array, Collection, or Map.

The *struts-html* custom tag library has been created (package `org.apache.struts.taglib.html`), based on the old tags that were related to HTML form presentation. The following differences from the old tags are notable:

- You must now reference the "struts-html.tld" TLD to access these tags.
- All attribute names matching JavaScript event handlers are now all lower case (onClick --> onclick) to conform to XHTML.
- The `options1` tag has been eliminated since Struts is now based on Java2.
- All tag implementation classes have had their `final` modifiers removed, and `private` instance variables changed to `protected`. This makes it possible to easily subclass these tags to provide specialized functionality.
- The `<html:link>` tag has been enhanced to support a new `page` attribute that allows you to use context-relative URIs in a portable manner.
- A new `<html:html>` tag has been created that renders an HTML `<html>` element with appropriate `lang` and `xml:lang` attributes, based on the locale stored for the user's session (if there is one).
- A new `<rewrite>` tag has been created that renders a request URI, possibly encoded with a session identifier, based on exactly the same rules used by the `<link>` tag that generates hyperlinks. These constants can be useful when you are creating JavaScript code that needs to be aware of Struts addressing concepts.
- The `options` tag now supports a new `collection` attribute, which can be used to specify a collection whose beans have properties that return the value (to be returned to the server) and the label (to be displayed to the user) from a single collection. The previous support for processing parallel collections is still available.
- The `form` tag has been enhanced to look up the name of the form bean, it's Java class,

and the scope in which the bean should be created or accessed, from the corresponding action mapping entry in the "struts-config.xml" file, if the name, scope, and type attributes are not specified. This removes the need to make changes in two places when these values are changed.

- A new `<image>` tag has been added, to create HTML input tags of type "image".
- The `form` tag has been enhanced to read its configuration from a corresponding action mapping entry in the "struts-config.xml" file, if the name, scope, and type attributes are not specified. It can look up the name of the form bean, its Java class, the scope in which the bean should be created or accessed, plus the path to which the form should be submitted. This removes the need to make changes in two places when these values are changed. It works for cases where the controller servlet is extension mapped or path mapped.
- The `<html:img>` tag has been added, to render an HTML `<img>` tag.

A new package of Actions and associated classes for handling file uploads has been created (package `org.apache.struts.upload`):

- The basic package of file upload handling actions has been created.
- An example application illustrating the use of the new features has been added (`webapps/struts-upload.war`).

The following major new features have been added to the utility classes library (package `org.apache.struts.util`):

- Initial implementation of a JDBC data source that implements the `javax.sql.DataSource` interface from the JDBC 2.0 Standard Extensions API. This implementation may be configured based on new extensions to the Struts configuration file DTD, and the configured data sources / connection pools are made available to application components as a servlet context attribute (i.e. an application scope bean).
- The previous implementation of `MessageResources`, which was ultimately based on `java.util.ResourceBundle`, has been completely replaced and re-implemented. The primary features of the new implementation are:
  - All components stored as servlet context attributes now implement the `java.io.Serializable` interface, to better integrate with application servers that prefer this.
  - The `MessageResources` and `MessageResourcesFactory` classes have been abstracted so that you can easily create your own implementations that derive their message strings from resources other than property files.
- Property gets and sets made through `PropertyUtils` can now use a new syntax for indexed and nested properties.
- Conversion to and from numeric types now support a configurable default value to use when conversion fails.

The following major new features have been added to the *Struts Example Application*:

- The form beans used in the example application now use request scope rather than session scope. This is the preferred approach for single page forms that contain *all* of your relevant properties, because there is no need to maintain such form beans across requests. Note that the action classes have been coded so that they work with either request scope or session scope beans.
- The Struts Example Application has been updated to utilize the new (separated) custom tag libraries, rather than the old combined one, as well as the latest features of the tags being used.
- A "Walking Tour of the Struts Example Application" has been added, to highlight features for newcomers to Struts.

The following major new documentation updates have been added to Struts:

- All documentation is generated from XML input files, using a standard stylesheet to create a common look and feel.
- Developer Guides for the Java classes in the `org.apache.struts.digester` and `org.apache.struts.util` packages.
- Developer Guides for the following Struts custom tag libraries have been added: `struts-bean`, `struts-html`, `struts-logic`, and `struts-template`.
- The Struts Users Guide has been brought up to date with respect to all of the changes since Struts 0.5, and separated into multiple HTML pages for easy reading.
- Installation information has been updated to include platform-specific notes, issues, and workarounds.
- A new resources page now points at external information and resources related to Struts.
- A new example application, `struts-blank`, is included as a quick starting point for new application development.

## Changes and Bug Fixes

The following changes and bug fixes to the controller servlet and associated classes (package `org.apache.struts.action`) have occurred:

- The `ActionMapping` interface has been converted to a base class instead, to reduce the impact of future enhancements. Anyone who has extended the `ActionMappingBase` convenience base class (which has been deprecated) should extend `ActionMapping` instead.
- In conjunction with the new configuration file format mentioned above, the properties of `ActionMapping` have been substantially updated. See the [API Documentation](#) for more information
- The `Action` interface has been converted to a base class instead, to reduce the impact of future enhancements. Anyone who has extended the `ActionBase` convenience base class (which has been deprecated) should extend `Action` instead.
- In conjunction with the above change, the `servlet` argument has been removed from the parameter list for the `perform()` method, because it is now redundant -- the associated servlet is set via the `setServlet()` method when the `Action` instance is first created.
- Responsibility for creating `Action` instances has been moved from `ActionMapping` to the controller servlet, so that instance lifecycle management can be performed. As a side effect of this change, if you had two actions that used the same `Action` class name, there will now be only one (shared) instance of the `Action` class, rather than two.
- New `log(String, int)` method that logs the associated message only if you have configured the debugging detail level for the servlet to an equal or higher value.
- In `ActionServlet`, the functionality to populate form bean parameters from a request, and the functionality to validate the form bean's contents, has been separated into two methods that can be overridden individually if required.
- The `ActionServlet` functionality to call the `validate()` method of a form bean is skipped if the user pressed the Cancel key (i.e. the submit button created by the `<html:cancel>` custom tag), or if the selected mapping does not define an input form to return control to.
- The controller servlet may now be used as the target of a `RequestDispatcher.include()` or `<jsp:include/>` call. Previously, it would mistakenly use the original request URI, rather than the included path, to calculate which action class to execute.
- The `ActionMappings.getUnknown()` method now takes the current request as an parameter, so that context-sensitive decisions can be made.



- When the controller servlet processes an `ActionForward` that has the `redirect` property set, it now performs URL rewriting to maintain session state even if cookies are not being used.
- The `ActionErrors` class now includes a method that will return an `Iterator` over the error messages related to a particular input field.

The following changes and bug fixes to the Digester module (package `org.apache.struts.digester`) have occurred:

- The `Digester.resolveEntity()` method has been enhanced to correctly handle local URIs so that it works with resources loaded via `Class.getResource()`.
- The input source handed to the Digester is now closed, even if a parsing exception is throw.

The following changes and bug fixes to the *struts-bean* custom tag library (package `org.apache.struts.taglib.bean`) have occurred:

- By default, the `<bean:write>` tag will filter output for characters sensitive to HTML. You can turn this off by adding a `filter="false"` attribute.
- When performing a `<bean:include>` in a page that is part of a session, pass the session identifier along on the generated request so that it will be part of the same session.
- The `<bean:define>` tag can now create beans directly from the `value` attribute, if desired.
- The `<bean:define>` tag now accepts an optional `toScope` attribute, to declare which scope the new bean should be created in. The default remains `page` scope.
- Default values can now be specified on `<bean:cookie>`, `<bean:header>`, and `<bean:parameter>` tags, which are used when the corresponding value is not present in the current request.

The following changes and bug fixes to the *struts-html* custom tag library (package `org.apache.struts.taglib.html`) have occurred:

- **WARNING** - When the Struts 0.5 tag library was originally split into separate libraries, this library was named *struts-form*, to reflect the fact that the majority of tags related to creating input forms. It has been renamed to *struts-html* to reflect the fact that all of the tags in this library are relevant only when building HTML-based user interfaces.
- The `<html:html>` tag now supports a `locale="true"` attribute that requests the same Locale negotiation (based on the presence of an `Accept-Language` header) that is performed by the controller servlet.
- The `<html:link>` tag now supports the ability to add a single request parameter (based on a bean property) in addition to the ability to add request parameters from a `Map`.
- The `<html:errors>` tag lets you select only the error messages related to a particular input field, or all errors.
- The `<html:password>` tag now optionally redisplay the previous value of the input field.
- The value returned by a `<html:multibox>` tag can now be specified in the body of the tag, as well as via the `value` attribute.

The following changes and bug fixes to the *struts-logic* custom tag library (package `org.apache.struts.taglib.logic`) have occurred:

- The `<logic:present>` and `<logic:notPresent>` tags now accept a `role` attribute that allows you to detect whether the currently authenticated user does, or does not, possess a particular security role.

The following changes and bug fixes to the *struts-template* custom tag library (package `org.apache.struts.taglib.template`) have occurred:

- The `<template:put>` tag now accepts a `direct` attribute that causes the content being put to be rendered directly, rather than being included.

The following changes and bug fixes to the Utilities package (package `org.apache.struts.util`) have occurred:

- Fixed `PropertyUtils.getPropertyType()` to correctly return the underlying element type even if there was a non-indexed property getter method.
- Added a missing "return" statement to `PropertyUtils.setIndexedProperty()`.
- Functionality in `BeanUtils` that previously duplicated functionality that was earlier moved to `PropertyUtils` has been removed.
- Fixed `PropertyUtils.copyProperties()` to skip cases where the origin bean has a getter method but the destination bean does not have a setter method.
- Added `BeanUtils.cloneBean()` to create a new instance of an existing bean, and copy all known properties, even if the bean class does not implement `Cloneable`.
- The `BeanUtils` class has been refactored so that it, and the associated `ConvertUtils` and `PropertyUtils` classes, can easily be used without having to have the servlet API classes available on the classpath.
- Property introspection is now smarter, so that you can access public methods declared in an implemented interface, even if the class itself is not public.

The following changes and bug fixes to the Struts Example Application (package `org.apache.struts.example` and the corresponding web components) have occurred:

- Used the `reset()` methods defined by the `ActionForm` interface to reset form bean properties to default values. This is particularly important to make boolean properties (represented visually by checkboxes) work correctly.
- Eliminate the special-case handling of null String values in the form beans. Such handling is not necessary because the custom tags correctly deal with null String values.
- Use the `PropertyUtils.copyProperties()` method to initially populate form beans from underlying data objects, and to update data objects when a transaction is successfully completed. Note that using this approach dramatically lessens an action class's dependence on the specific properties of the form bean and corresponding data object in many use cases.
- Added an `autoConnect` boolean property to the Subscription data object, primarily to illustrate that representing a boolean property with a checkbox now works correctly if you set the property to `false` in the `reset()` method of your form beans.



## Technologies

## Downloads

## Documentation

## Spotlight

## Developer Services

## Java BluePrints

## JavaServer Pages

## Documentation

## Downloads

## Industry Momentum

## FAQ

## Tag Libraries

## JSTL

## JavaServer Faces

## Resources

## J2EE Home


**Tomcat**  
**@ JAKARTA**
**Printable Page**

## JavaServer Pages™

### Standard Tag Library

The JavaServer Pages Standard Tag Library (JSTL) encapsulates, as simple tags, core functionality common to many JSP applications. For example, instead of suggesting that you iterate over lists using a scriptlet or different iteration tags from numerous vendors, JSTL defines a standard tag that works the same everywhere. This standardization lets you learn a single tag and use it on multiple JSP containers. Also, when tags are standard, containers can recognize them and optimize their implementations.

JSTL has support for common, structural tasks such as iteration and conditionals, tags for manipulating XML documents, internationalization and locale-sensitive formatting tags, and SQL tags. It also introduces a new expression language to simplify page development, and it provides an API for developers to simplify the configuration of JSTL tags and the development of custom tags that conform to JSTL conventions.

### Tutorials

- [Java Web Services Tutorial](#) for the [Java Web Services Developer Pack](#)

### Reference Summary

- [JSTL 1.0](#) (from *JSTL in Action*, Manning)

### Articles

- [JSTL 1.0: What JSP Applications Need, Part 2](#), *OnJava*, September 11, 2002
- [Standardizing JSP, Part 1](#), *OnJava*, August 14, 2002
- [JSP Standard Tag Libraries, Part 2](#), *OnJava*, May 8, 2002
- [JSP Standard Tag Libraries, Part 1](#), *OnJava*, March 13, 2002

### Books

- [JSTL in Action](#), Shawn Bayern, Manning ([examples](#))
- [JavaServer Pages](#), Hans Bergsten, O'Reilly
- [Core JSTL: Mastering the JSP Standard Tag](#)

### News & Highlights

**NEW** [Sun ONE Studio 4](#) contains JSTL 1.0.

**NEW** Java Web Services Developer Pack: [Java WSDP 1.0](#) is now generally available. Java WSDP 1.0 includes the latest implementations of JSTL 1.0, JSP 1.2, and Servlet 2.3 and is a great vehicle for developing Web applications as well as Web services.

[Library](#), David Geary, Sun Microsystems Press

## Downloads

## IMPLEMENTATIONS

### Java Web Services Developer Pack

[Java Web Services Developer Pack](#) (Java WSDP) is an integrated toolset that allows Java developers to build, test, and deploy XML applications, Web services, and Web applications. The Java WSDP includes the reference implementation of JSTL 1.0.

#### Java WSDP 1.0

[Download](#)

### Jakarta Taglibs

[Jakarta Taglibs](#) is a free, open-source implementation of tag libraries developed under the Jakarta project at the Apache Software Foundation.

#### Standard 1.0

The Jakarta Taglibs [Standard 1.0](#) tag library is an implementation of JSTL 1.0.

## SPECIFICATION

### JavaServer Pages Standard Tag Library Specification

The JavaServer Pages Standard Tag Library specification was created under the Java Community Process to provide full public participation in its definition and development.

#### 1.0 - Final Release

|                  |  |
|------------------|--|
| Specification    | <a href="#">Download</a>   |
| Send comments to | <a href="mailto:jsr-52-comments@jcp.org">jsr-52-comments@jcp.org</a> |

[ This page was last updated Sep-27-2002 ]

[Company Info](#) | [Licensing](#) | [Employment](#) | [Press](#) | [Help](#) |  
[JavaOne](#) | [Java Community Process](#) | [Java Wear and Books](#)



Unless otherwise licensed, code in all technical manuals herein (including articles, FAQs, samples) is provided under this [License](#).

Copyright © 1995-2003 [Sun Microsystems, Inc.](#)  
All Rights Reserved. [Terms of Use](#). [Privacy Policy](#).

# Package org.apache.struts.taglib.bean

The "struts-bean" tag library contains JSP custom tags useful in defining new beans (in any desired scope) from a variety of possible sources, as well as a tag to render a particular bean (or bean property) to the output response.

See: [Description](#)

| Class Summary                |   |
|------------------------------|---|
| <a href="#">CookieTag</a>    | Define a scripting variable based on the value(s) of the specified cookie received with this request.   |
| <a href="#">CookieTei</a>    | Implementation of TagExtraInfo for the <b>cookie</b> tag, identifying the scripting object(s) to be made visible.   |
| <a href="#">DefineTag</a>    | Define a scripting variable based on the value(s) of the specified bean property.   |
| <a href="#">DefineTei</a>    | Implementation of TagExtraInfo for the <b>define</b> tag, identifying the scripting object(s) to be made visible.   |
| <a href="#">HeaderTag</a>    | Define a scripting variable based on the value(s) of the specified header received with this request.   |
| <a href="#">HeaderTei</a>    | Implementation of TagExtraInfo for the <b>header</b> tag, identifying the scripting object(s) to be made visible.   |
| <a href="#">IncludeTag</a>   | Define the contents of a specified intra-application request as a page scope attribute of type java.lang.String.  |
| <a href="#">IncludeTei</a>   | Implementation of TagExtraInfo for the <b>include</b> tag, identifying the scripting object(s) to be made visible.  |
| <a href="#">MessageTag</a>   | Custom tag that retrieves an internationalized messages string (with optional parametric replacement) from the ActionResources object stored as a context attribute by our associated ActionServlet implementation. |
| <a href="#">PageTag</a>      | Define a scripting variable that exposes the requested page context item as a scripting variable and a page scope bean.   |
| <a href="#">PageTei</a>      | Implementation of TagExtraInfo for the <b>page</b> tag, identifying the scripting object(s) to be made visible.   |
| <a href="#">ParameterTag</a> | Define a scripting variable based on the value(s) of the specified parameter received with this request.  |
| <a href="#">ParameterTei</a> | Implementation of TagExtraInfo for the <b>parameter</b> tag, identifying the scripting object(s) to be made visible.  |
| <a href="#">ResourceTag</a>  | Define a scripting variable based on the contents of the specified web application resource.  |
| <a href="#">ResourceTei</a>  | Implementation of TagExtraInfo for the <b>resource</b> tag, identifying the scripting object(s) to be made visible.   |
| <a href="#">SizeTag</a>      | Define a scripting variable that will contain the number of elements found in a specified array, Collection, or Map.  |

|                           |   |
|---------------------------|---|
| <a href="#">SizeTei</a>   | Implementation of TagExtraInfo for the <b>size</b> tag, identifying the scripting object(s) to be made visible.   |
| <a href="#">StrutsTag</a> | Define a scripting variable that exposes the requested Struts internal configuraton object.   |
| <a href="#">StrutsTei</a> | Implementation of TagExtraInfo for the <b>struts</b> tag, identifying the scripting object(s) to be made visible.   |
| <a href="#">WriteTag</a>  | Tag that retrieves the specified property of the specified bean, converts it to a String representation (if necessary), and writes it to the current output stream, optionally filtering characters that are sensitive in HTML. |

## Package org.apache.struts.taglib.bean Description

The "struts-bean" tag library contains JSP custom tags useful in defining new beans (in any desired scope) from a variety of possible sources, as well as a tag to render a particular bean (or bean property) to the output response.

[\[Introduction\]](#) [\[Bean Properties\]](#) [\[Bean Creation\]](#) [\[Bean Output\]](#)

### Introduction

Much of the power of JavaServer Pages (JSP) technology comes from the simple and powerful mechanisms by which the servlet that is generated automatically from your JSP source page can interact with JavaBeans that represent the computational state of your application. In standard JSP pages, the `<jsp:useBean>` tag is used create a bean (if necessary), as well as a "scripting variable" that can be used within scriptlets to refer to these beans.

The "struts-bean" tag library provides substantial enhancements to the basic capability provided by `<jsp:useBean>`, as discussed in the following sections:

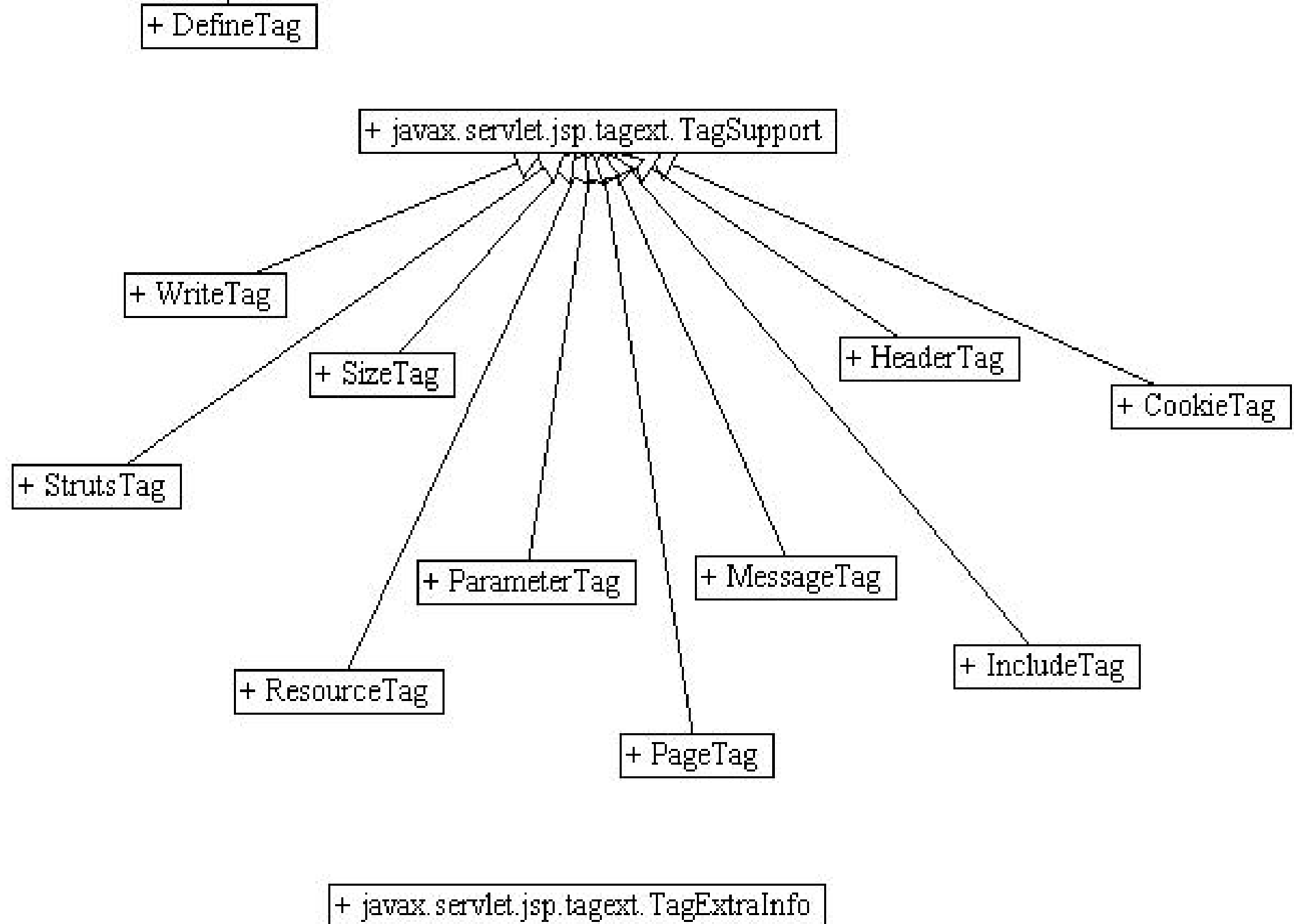
- [Bean Properties](#) - Extended syntax to refer to JavaBean properties with simple names (same as the standard JSP tags `<jsp:getProperty>` and `<jsp:setProperty>`), nested names (a property named `address.city` returns the value retrieved by the Java expression `getAddress().getCity()`), and indexed names (a property named `address[3]` retrieves the fourth address from the indexed "address" property of a bean).
- [Bean Creation](#) - New JSP beans, in any scope, can be created from a variety of objects and APIs associated with the current request, or with the servlet container in which this page is running.
- [Bean Output](#) - Supports the rendering of textual output from a bean (or bean property), which will be included in the response being created by your JSP page.

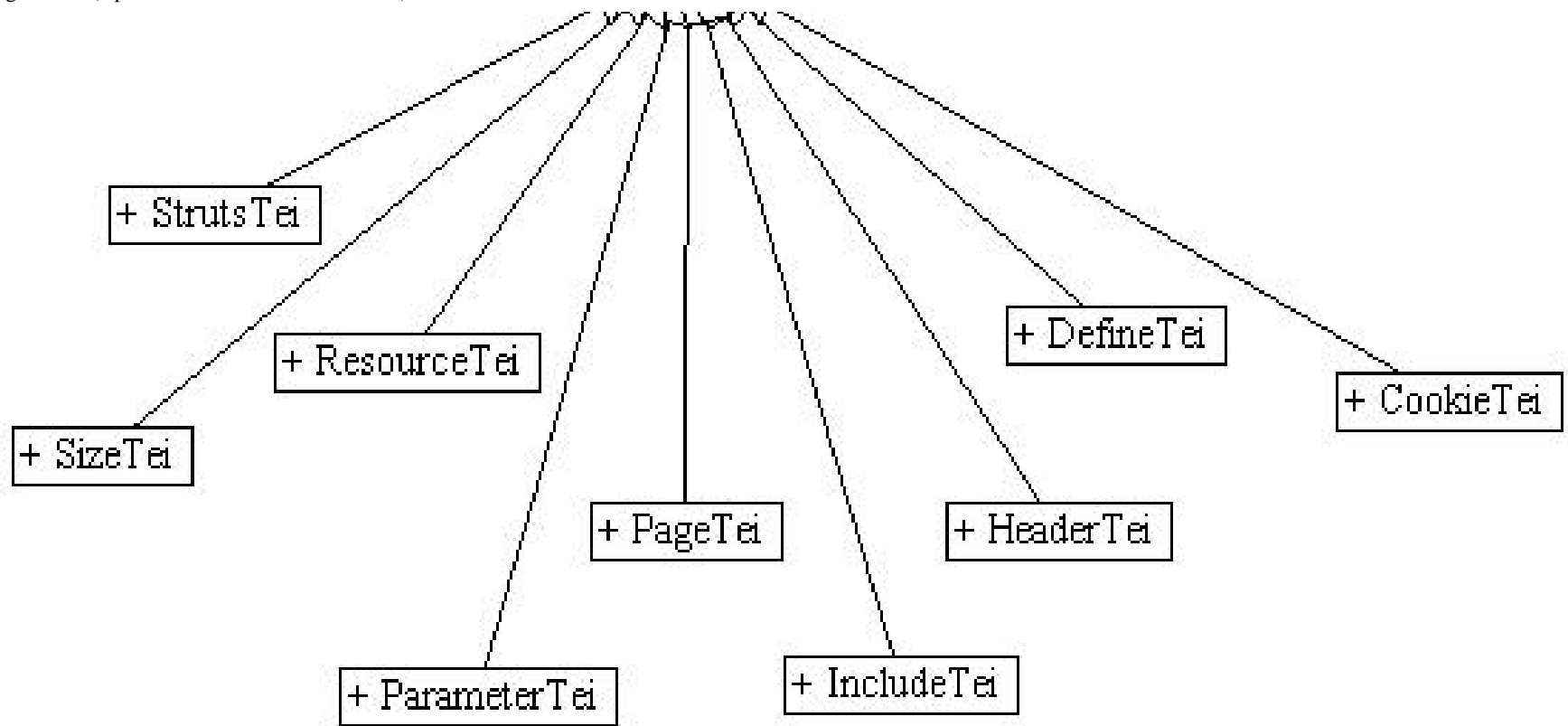
See the [Bean Tags Reference](#) for detailed information about the available tags in this tag library, and the valid attributes for each tag.

+ javax.servlet.jsp.tagext.BodyTagSupport



|  |  |   |                     |
|--|--|---|---------------------|
|  |  | <b>Project:</b> Struts                        | <b>Version:</b> 1.1 |
|  |  | <b>Package:</b> org.apache.struts.taglib.bean | <b>Version:</b> 1.1 |

**Date:** 29 October 2002**Description:**



## Bean Properties

### Common Tag Attributes

The tags in the "struts-bean" tag library (and, generally, in all tag libraries included with the Struts framework) share a common set of tag attributes that have the same meaning, no matter what tag they are used on. These common attributes include:

- *id* - Names the scripting variable that will be created by this custom tag, as well as the key value used to locate this bean in the scope defined by the *scope* attribute.
- *name* - Defines the key value by which an existing bean will be looked up in the scope defined by the *scope* attribute (if any), or by searching through the various scopes in the standard order (page, request, session, application).
- *property* - Defines the name of a JavaBeans property, of the JSP bean identified by the *name* and (optional) *scope* attributes, whose value is to be used by this custom tag. If not specified, the bean identified by *name* is itself used as the value of interest. See below for more discussion about how a property can be referenced.
- *scope* - Identifies the JSP scope ("page", "request", "session", or "application" within which a particular bean will be searched for (under the key specified by the *name* attribute) or created (under the key specified by the *id* attribute). If not specified, beans will generally be searched for in the order listed above, or created in page scope.



## Property References

Struts tags that support the `property` tag generally also recognize a rich syntax for getting and setting properties. There are three types of references supported: simple, nested, and indexed.

*Simple References* - These are equivalent to the syntax you use with the standard `<jsp:getProperty>` and `<jsp:setProperty>` tags. A reference to a property named "foo" is converted into a method call to `getFoo()` or `setFoo(value)` (as appropriate), using the standard JavaBeans Specification naming conventions for bean properties. Struts uses the standard Java introspection APIs to identify the names of the actual property getter and setter methods, so your beans can provide customized method names through the use of a `BeanInfo` class. See the JavaBeans Specification, available at <http://java.sun.com/products/javabeans/>, for more information.

*Nested References* - Nested references are used to access a property through a hierarchy of property names separated by periods ("."), similar to the way that nested properties are accessed in JavaScript. For example, the following property reference in a getter (such as the `<bean:define>` tag discussed below):

```
property="foo.bar.baz"
```

is translated into the equivalent the Java expression:

```
getFoo().getBar().getBaz()
```

If a nested reference is used in a setter (such as when an input form is processed), the property setter is called on the **last** property in the chain. For the above property reference, the equivalent Java expression would be:

```
getFoo().getBar().setBaz(value)
```

*Indexed References* - Subscripts can be used to access individual elements of properties whose value is actually an array, or whose underlying JavaBean offers indexed getter and setter methods. For example, the following property reference in a getter (such as the `<bean:define>` tag discussed below):

```
property="foo[2]"
```

is translated into the equivalent of the Java expression:

```
getFoo(2);
```

while the same property reference in a setter would call the equivalent of:

```
setFoo(2, value)
```

As you can see from the above translations, the subscripts used in indexed references are **zero relative** (that is, the first element in an array is `foo[0]`), just as is true in the Java language.

*Combined References* - Nesting and indexing can be combined in arbitrary ways, so that expressions like `foo.bar[0].baz[2]` are legal. You must be careful, of course, to ensure that the actual beans being accessed by these references have properties of the appropriate names and types. Otherwise, JSP runtime exceptions will be thrown.

See the JavaDocs for [PropertyUtils](#) for more detailed information about the mechanisms that Struts uses to access properties in a general way, through Java reflection APIs.

---

## Bean Creation

### Introduction

New beans can be created, and introduced into one of the four standard JSP scopes (page, request, session, and application) through a variety of techniques. The following subsections describe the use of the following approaches:

- Java Code in Action Classes
- Java Code in Scriptlets
- The Standard `<jsp:useBean>` Tag
- The Struts `<bean:define>` Tag
- Other Struts Copying Tags

### Java Code in Action Classes

Because the JSP pages are compiled into Servlets, your `Action` classes that are invoked by the Struts controller servlet have convenient access to three of the four standard JSP scopes (request, session, and application). It is very common practice for the business logic contained in your `Action` class to create results that are stored in request or session scope, which will be used by a JSP page you forward control to in rendering the next page of the user interface.

*Request Scope* - To store a bean in request scope under name "cust", your `Action` class would execute code similar to this:

```
Customer customer = ... create or acquire a customer reference ...;
request.setAttribute("cust", customer);
```

*Session Scope* - To store a bean in session scope under name "user" (perhaps in a logon action), your `Action` class would execute code similar to this:

```
User user = ... look up valid user in the database ...;
HttpSession session = request.getSession();
session.setAttribute("user", user);
```

*Application Scope* - Generally, application scope beans are initialized in the `init()` method of a startup servlet. However, it is legal for an `Action` class to create such beans, if this is appropriate, like this:

```
Foo foo = ... create a Foo ...;
servlet.getContext().setAttribute("foo", foo);
```

### Java Code in Scriptlets

While it is not a recommended practice in Struts-based applications (because developers will be tempted to mix business logic and presentation logic in their JSP pages), it is legal for scriptlet code in a JSP page to create new JavaBeans dynamically, and add them to any of the four possible scopes, as demonstrated in the code examples below:

*Page Scope* - To store a bean in page scope under name "foo", your scriptlet must execute code like this:

```
<%
    Foo foo = ... create a foo ...;
    pageContext.setAttribute("foo", foo, PageContext.PAGE_SCOPE);
%>
```

*Request Scope* - To store a bean in request scope under name "cust", your scriptlet must execute code like this:

```
<%
    Customer customer = ... create or acquire a customer reference ...;
    pageContext.setAttribute("cust", customer, PageContext.REQUEST_SCOPE);
%>
```

*Session Scope* - To store a bean in session scope under name "user", (perhaps as a result of a validated login), your scriptlet must execute code like this:

```
<%
    User user = ... look up valid user in the database ...;
    pageContext.setAttribute("user", user, PageContext.SESSION_SCOPE);
%>
```

*Application Scope* - Generally, application scope beans are initialized in the `init()` method of a startup servlet. However, a scriptlet can create such beans, if appropriate, like this:

```
<%
    Foo foo = ... create a Foo ...;
    pageContext.setAttribute("foo", foo, PageContext.APPLICATION_SCOPE);
%>
```

**NOTE** - As mentioned above, using scriptlets in your JSP pages is strongly discouraged in a Struts based application, unless you are executing code that is **only** related to presentation of existing data. In general, your application's processing logic should be encapsulated in `Action` classes (or in beans or EJBs called by those classes), rather than being intermixed in your JSP pages.

## The Standard `<jsp:useBean>` Tag

JavaServer Pages (JSP) offers a standard tag, `<jsp:useBean>` that can be used to create a new bean, or introduce a reference to an existing bean, into a JSP page. Beans (or bean references) introduced through this mechanism are completely interoperable with beans created by any of the Struts creation techniques described in this section.

You **must** use `<jsp:useBean>` to introduce a reference to an existing bean, if you wish to reference that bean with other standard JSP tags (such as `<jsp:getProperty>` or `<jsp:setProperty>`). If you only wish to reference such beans with other Struts tags, use of `<jsp:useBean>` is not required.

For more information about the `<jsp:useBean>` tag, see the JavaServer Pages Specification, available at <http://java.sun.com/products/jsp/download.html> .

## The Struts `<bean:define>` Tag

Struts provides a powerful, general purpose, tag (`<bean:define>` ) that can be used to create a new bean, in any scope, by copying another bean (or the value of the property of another bean). This tag supports the "property" attribute, and therefore all the power of property references, as discussed [above](#) . It can be used in a variety of different ways, described further below. Unless you specify the "toScope" attribute, all defined beans will be created in page scope.

*Introduce A String Constant* - You can create a new bean that has a constant String value (or the result of calculating a runtime expression):

```
<bean:define id="foo" value="This is a new String"/>
<bean:define id="bar" value='<%= "Hello, " + user.getName() %>' />
<bean:define id="last" scope="session"
    value='<%= request.getRequestURI() %>' />
```

*Copy An Existing Bean* - You can create a new reference to an existing bean object. You can specify the Java class or interface the new bean is expected to conform to with the "type" attribute, or accept the default type of `java.lang.Object` (this only affects the scripting variable that is exposed to scriptlets, so it is not generally meaningful in Struts-based applications).

```
<bean:define id="foo" name="bar" />
<bean:define id="baz" name="bop" type="com.mycompany.MyBopClass" />
```

*Copy An Existing Bean Property* - You can create a new bean that is initialized to the value returned by a property getter. The value of the "property" attribute can be any simple, nested, or indexed property reference that follows the rules described earlier. In the first example below, we also illustrate accessing the property of a request scope bean, and creating the new bean in session scope (rather than the default page scope).

```
<bean:define id="foo" name="bar" property="baz" scope="request"
    toScope="session" />
<bean:define id="bop" name="user" property="role[3].name" />
```

## Other Struts Copying Tags

Struts offers a variety of bean creation tags that copy existing beans (or bean properties) from the environment within which this page is running, and the request that is currently being processed. Not all of the attributes for each tag are illustrated in the examples below - see the [Bean Tags Reference](#) for more information. Any bean created by these tags exists only in page scope, for the remainder of the current page.

*Copy A Cookie* - You can create a new bean containing a `javax.servlet.http.Cookie` that was included in the current request. If no cookie of the specified name was included, a request time exception will be thrown - therefore, it is common to nest the use of this tag inside a `<logic:present cookie="xxx">` tag to ensure that the cookie was really included. If there is the possibility that more than one cookie of the same name was included, specify the "multiple" attribute (and the resulting bean will be an array of Cookies, instead of a single Cookie).

```
<bean:cookie id="foo" name="cookiesname" />
<bean:cookie id="all" name="JSESSIONID" multiple="true" />
```

*Copy A Request Header* - You can create a new bean containing the value of an HTTP header included in this request. If no header of the specified name was included, a request time exception will be thrown - therefore, it is common to nest the use of this tag inside a `<logic:present header="xxx">` tag to ensure that the header was really included. If there is the possibility that more than one header of the same name was included, specify the "multiple" attribute (and the resulting value bean will be an array of String values, instead of a single String).

```
<bean:header id="agent" name="User-Agent" />
<bean:header id="languages" name="Accept-Language" multiple="true" />
```

*Copy A Dynamically Created Response* - You can generate an internal request to the application you are running, and turn the response data that is returned from that request into a bean (of type String). One possible use for this technique is to acquire dynamically created XML formatted data that will be stored in a bean and later manipulated (such as by applying an XSLT stylesheet). If the current request is part of a session, the generated request for the include will also include the session identifier (and thus be considered part of the same session).

```
<bean:include id="text" name="/generateXml?param1=a&param2=b" />
```

*Copy A JSP Implicitly Defined Object* - You can create a bean that is one of the JSP implicitly defined objects (see the JSP spec for more details). This is useful if you wish to perform property getter actions against the implicit object with a custom tag instead of a scriptlet.

```
<bean:page id="app" property="application" />
<bean:page id="sess" property="session" />
```

*Copy A Request Parameter* - You can create a new bean containing the value of a parameter included in this request. If no parameter of the specified name was included, a request time exception will be thrown - therefore, it is common to nest the use of this tag inside a `<logic:present parameter="xxx">` tag to ensure that the parameter was really included. If there is the possibility that more than one parameter of the same name was included, specify the "multiple" attribute (and the resulting value bean will be an array of String values, instead of a single String).

```
<bean:parameter id="name" name="name" />
<bean:header id="options" name="option" multiple="true" />
```

*Copy a Web Application Resource* - You can create a new bean containing either the value of a web application resource as a String, or a `java.io.InputStream` for reading the content of that resource. The resource is accessed with a context-relative path (beginning with "/"), using the `ServletContext.getResource()` or `ServletContext.getResourceAsStream()` methods on the underlying application object.

```
<bean:resource id="deployment" name="/WEB-INF/web.xml"/>
<bean:resource id="stream" name="/WEB-INF/web.xml"
    input="true"/>
```

*Copy A Struts Configuration Object* - You can create a new bean containing one of the standard Struts framework configuration objects. Doing this gives you access to the properties of the configuration object, if needed.

```
<bean:struts id="form" formBean="CustomerForm"/>
<bean:struts id="fwd" forward="success"/>
<bean:struts id="map" mapping="/saveCustomer"/>
```

---

## Bean Output

None of the Struts Bean tags discussed so far render any output to the response page that is being generated from this JSP page. They are executed in order to make relevant Java objects visible as beans for further manipulation. The following tags cause output to be written to the response, and therefore made visible to the ultimate requester.

*Render An Internationalized Message* - You can specify a message key (with optional parameter replacement objects) that are passed to a [MessageResources](#) object that returns the corresponding message text. The message text will be copied to the response currently being created. By default, messages are looked up in the application resources bundle that is initialized for you (as an application scope bean) by the Struts controller servlet, using the Locale must recently stored in the user's session. These defaults can be overridden by setting values for the "bundle" and "locale" attributes, as described in the [Bean Tags Reference](#).

```
<bean:message key="label.Cancel"/>
<bean:message key="message.hello" arg0='<%= user.getFullName() %>'/>
```

*Render A Bean or Bean Property* - The contents of a bean, or bean property, are converted to a String and then copied to the response currently being created. This tag understands the syntax for simple, nested, and indexed property references described [above](#). Beans from any scope can be requested - by default, the scopes are searched in expanding visibility order (page, request, session, and application) to locate the requested bean.

```
<bean:write name="username"/>
<bean:write name="user" property="fullName"/>
<bean:write name="customer" property="orders[2].partNumber"
    scope="session"/>
```

[Overview](#) **Package** [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV PACKAGE](#) [NEXT PACKAGE](#)

[FRAMES](#)

[NO FRAMES](#)

[All Classes](#)

---

Copyright © 2000-2002 - Apache Software Foundation



## User Guide

[Table of Contents](#)

[Preface](#)

[Introduction](#)

[Model](#)

[Components](#)

[View](#)

[Components](#)

[Controller](#)

[Components](#)

[Configuration](#)

[Release Notes](#)

[Installation](#)

## Struts Bean Tags

This tag library contains tags useful in accessing beans and their properties, as well as defining new beans (based on these accesses) that are accessible to the remainder of the page via scripting variables and page scope attributes. Convenient mechanisms to create new beans based on the value of request cookies, headers, and parameters are also provided.

Many of the tags in this tag library will throw a `JspException` at runtime when they are utilized incorrectly (such as when you specify an invalid combination of tag attributes). JSP allows you to declare an "error page" in the `<%@ page %>` directive. If you wish to process the actual exception that caused the problem, it is passed to the error page as a request attribute under key `org.apache.struts.action.EXCEPTION`.

If you are viewing this page from within the Struts Documentation Application (or online at <http://jakarta.apache.org/struts/>), you can learn more about using these tags in the [Bean Tags Developer's Guide](#).

## Developer Guides

[Bean Tags](#)

[HTML Tags](#)

[Logic Tags](#)

[Nested Tags](#)

[Template Tags](#)

[Tiles Tags](#)

[Utilities](#)

[Validator](#)

## Quick Links

[Welcome](#)

[News and Status](#)

[Resources](#)

[User and  
Developer  
Guides \\*](#)

[FAQs and  
HowTos](#)

| Tag Name                  | Description   |
|---------------------------|---|
| <a href="#">cookie</a>    | Define a scripting variable based on the value(s) of the specified request cookie.    |
| <a href="#">define</a>    | Define a scripting variable based on the value(s) of the specified bean property.     |
| <a href="#">header</a>    | Define a scripting variable based on the value(s) of the specified request header.    |
| <a href="#">include</a>   | Load the response from a dynamic application request and make it available as a bean. |
| <a href="#">message</a>   | Render an internationalized message string to the response.                           |
| <a href="#">page</a>      | Expose a specified item from the page context as a bean.                              |
| <a href="#">parameter</a> | Define a scripting variable based on the value(s) of the specified request parameter. |
| <a href="#">resource</a>  | Load a web application resource and make it available as a bean.                      |
| <a href="#">size</a>      | Define a bean containing the number of elements in a Collection or Map.               |
| <a href="#">struts</a>    | Expose a named Struts internal configuration object as a bean.                        |
| <a href="#">write</a>     | Render the value of the specified bean property to the current JspWriter.             |

**cookie** - Define a scripting variable based on the value(s) of the specified request cookie.



Retrieve the value of the specified request cookie (as a single value or multiple values, depending on the `multiple` attribute), and define the result as a page scope attribute of type `Cookie` (if `multiple` is not specified) or `Cookie[]` (if `multiple` is specified).

If no cookie with the specified name can be located, and no default value is specified, a request time exception will be thrown.

| Attribute Name        | Description   |
|-----------------------|---|
| <code>id</code>       | Specifies the name of the scripting variable (and associated page scope attribute) that will be made available with the value of the specified request cookie.<br><br>(REQUIRED) (RT EXPR)  |
| <code>multiple</code> | If any arbitrary value for this attribute is specified, causes all matching cookies to be accumulated and stored into a bean of type <code>Cookie[]</code> . If not specified, the first value for the specified cookie will be retrieved as a value of type <code>Cookie</code> .<br><br>(RT EXPR) |
| <code>name</code>     | Specifies the name of the request cookie whose value, or values, is to be retrieved.<br><br>(REQUIRED) (RT EXPR)  |
| <code>value</code>    | The default cookie value to return if no cookie with the specified name was included in this request.<br><br>(RT EXPR)  |

[Back to top](#)

**define** - Define a scripting variable based on the value(s) of the specified bean property.

Create a new attribute (in the scope specified by the `toScope` property, if any), and a corresponding scripting variable, both of which are named by the value of the `id` attribute. The corresponding value to which this new attribute (and scripting variable) is set are specified via use of exactly one of the following approaches (trying to use more than one will result in a `JspException` being thrown):

- Specify a `name` attribute (plus optional `property` and `scope` attributes) - The created attribute and scripting variable will be of the type of the retrieved JavaBean property, unless it is a Java primitive type, in which case it will be wrapped in the appropriate wrapper class (i.e. `int` is wrapped by `java.lang.Integer`).
- Specify a `value` attribute - The created attribute and scripting variable will be of type `java.lang.String`, set to the value of this attribute.
- Specify nested body content - The created attribute and scripting variable will be of type `java.lang.String`, set to the value of the nested body content.

If a problem occurs while retrieving the specified bean property, a request time exception will be thrown.

The `<bean:define>` tag differs from `<jsp:useBean>` in several ways, including:

- Unconditionally creates (or replaces) a bean under the specified identifier.
- Can create a bean with the value returned by a property getter of a different bean (including properties referenced with a nested and/or indexed property name).
- Can create a bean whose contents is a literal string (or the result of a runtime expression) specified by the `value` attribute.
- Does not support nested content (such as `<jsp:setProperty>` tags) that are only executed if a bean was actually created.

**USAGE NOTE** - There is a restriction in the JSP 1.1 Specification that disallows using the same value for an `id` attribute more than once in a single JSP page. Therefore, you will not be able to use `<bean:define>` for the same bean name more than once in a single page.

See the Bean Developer's Guide section on [bean creation](#) for more information about these differences, as well as alternative approaches to introducing beans into a JSP page.

| Attribute Name | Description   |
|----------------|---|
| id             | Specifies the name of the scripting variable (and associated page scope attribute) that will be made available with the value of the specified property.<br><br>(REQUIRED) (RT EXPR)  |
| name           | Specifies the attribute name of the bean whose property is accessed to define a new page scope attribute (if <code>property</code> is also specified) or the attribute name of the bean that is duplicated with the new reference created by this tag (if <code>property</code> is not also specified). This attribute is required unless you specify a <code>value</code> attribute or nested body content.<br><br>(RT EXPR) |
| property       | Specifies the name of the property to be accessed on the bean specified by <code>name</code> . This value may be a simple, indexed, or nested property reference expression. If not specified, the bean identified by <code>name</code> is given a new reference identified by <code>id</code> .<br><br>(RT EXPR)   |
| scope          | Specifies the variable scope searched to retrieve the bean specified by <code>name</code> . If not specified, the default rules applied by <code>PageContext.findAttribute()</code> are applied.<br><br>(RT EXPR)   |
| toScope        | Specifies the variable scope into which the newly defined bean will be created. If not specified, the bean will be created in page scope.<br><br>(RT EXPR)  |
| type           | Specifies the fully qualified class name of the value to be exposed as the <code>id</code> attribute.<br><br>[ <code>java.lang.String</code> (if you specify a <code>value</code> attribute) or <code>java.lang.Object</code> otherwise. ] (RT EXPR)  |

|       |  |
|-------|--|
| value | <p>The <code>java.lang.String</code> value to which the exposed bean should be set. This attribute is required unless you specify the <code>name</code> attribute or nested body content.</p> <p>(RT EXPR)</p> |
|-------|--|

[Back to top](#)

#### **header** - Define a scripting variable based on the value(s) of the specified request header.

Retrieve the value of the specified request header (as a single value or multiple values, depending on the `multiple` attribute), and define the result as a page scope attribute of type `String` (if `multiple` is not specified) or `String[]` (if `multiple` is specified).

If no header with the specified name can be located, and no default value is specified, a request time exception will be thrown.

| Attribute Name | Description  |
|----------------|--|
| id             | <p>Specifies the name of the scripting variable (and associated page scope attribute) that will be made available with the value of the specified request header.</p> <p>(REQUIRED) (RT EXPR)</p>  |
| multiple       | <p>If any arbitrary value for this attribute is specified, causes a call to <code>HttpServletRequest.getHeaders()</code> and a definition of the result as a bean of type <code>String[]</code>. Otherwise, <code>HttpServletRequest.getHeader()</code> will be called, and a definition of the result as a bean of type <code>String</code> will be performed.</p> <p>(RT EXPR)</p> |
| name           | <p>Specifies the name of the request header whose value, or values, is to be retrieved.</p> <p>(REQUIRED) (RT EXPR)</p>  |
| value          | <p>The default header value to return if no header with the specified name was included in this request.</p> <p>(RT EXPR)</p>  |

[Back to top](#)

#### **include** - Load the response from a dynamic application request and make it available as a bean.

Perform an internal dispatch to the specified application component (or external URL) and make the response data from that request available as a bean of type `String`. This tag has a function similar to that of the standard `<jsp:include>` tag, except that the response data is stored in a page scope attribute instead of being written to the output stream. If the current request is part of a session, the generated request for the include will also include the session identifier (and thus be part of the same session).

The URL used to access the specified application component is calculated based on which of the following attributes you specify (you must specify exactly one of them):

- *forward* - Use the value of this attribute as the name of a global `ActionForward` to be looked up, and use the application-relative or context-relative URI found there.
- *href* - Use the value of this attribute unchanged (since this might link to a resource external to the application, the session identifier is **not** included).
- *page* - Use the value of this attribute as an application-relative URI to the desired resource.

| Attribute Name | Description   |
|----------------|---|
| anchor         | Optional anchor tag ("#xxx") to be added to the generated hyperlink. Specify this value <b>without</b> any "#" character.<br><br>(RT EXPR)  |
| forward        | Logical name of a global <code>ActionForward</code> that contains the actual content-relative URI of the resource to be included.<br><br>(RT EXPR)  |
| href           | Absolute URL (including the appropriate protocol prefix such as "http:") of the resource to be included. Because this URL could be external to the current web application, the session identifier will <b>not</b> be included in the request.<br><br>(RT EXPR) |
| id             | Specifies the name of the scripting variable (and associated page scope attribute) that will be made available with the value of the specified web application resource.<br><br>(REQUIRED) (RT EXPR)  |
| name           | Application-relative name (starting with a '/') of the web application resource to be dispatched, and whose response data is to be made available as a bean.<br><b>DEPRECATED - Use the "page" attribute instead.</b><br><br>(RT EXPR)                          |
| page           | Application-relative URI (starting with a '/') of the web application resource to be included.<br><br>(RT EXPR)   |
| transaction    | Set to <code>true</code> if you want the current transaction control token included in the generated URL for this include.<br><br>(RT EXPR)   |

[Back to top](#)**message** - Render an internationalized message string to the response.

Retrieves an internationalized message for the specified locale, using the specified message key, and write it to the output stream. Up to five parametric replacements (such as "{0}") may be specified.

The message key may be specified directly, using the `key` attribute, or indirectly, using the `name` and `property` attributes to obtain it from a bean.

| Attribute Name      | Description  |
|---------------------|--|
| <code>arg0</code>   | First parametric replacement value, if any.<br>(RT EXPR)   |
| <code>arg1</code>   | Second parametric replacement value, if any.<br>(RT EXPR)  |
| <code>arg2</code>   | Third parametric replacement value, if any.<br>(RT EXPR)   |
| <code>arg3</code>   | Fourth parametric replacement value, if any.<br>(RT EXPR)  |
| <code>arg4</code>   | Fifth parametric replacement value, if any.<br>(RT EXPR)   |
| <code>bundle</code> | The name of the application scope bean under which the <code>MessageResources</code> object containing our messages is stored.<br><br>[Action.MESSAGES_KEY] (RT EXPR)  |
| <code>key</code>    | The message key of the requested message, which must have a corresponding value in the message resources. If not specified, the key is obtained from the <code>name</code> and <code>property</code> attributes.<br><br>(RT EXPR)  |
| <code>locale</code> | The name of the session scope bean under which our currently selected <code>Locale</code> object is stored.<br><br>[Action.LOCALE_KEY] (RT EXPR)   |
| <code>name</code>   | Specifies the attribute name of the bean whose property is accessed to retrieve the value specified by <code>property</code> (if specified). If <code>property</code> is not specified, the value of this bean itself will be used as the message resource key.<br><br>(RT EXPR) |

|          |   |
|----------|---|
| property | Specifies the name of the property to be accessed on the bean specified by name. This value may be a simple, indexed, or nested property reference expression. If not specified, the value of the bean identified by name will itself be used as the message resource key.<br><br>(RT EXPR) |
| scope    | Specifies the variable scope searched to retrieve the bean specified by name. If not specified, the default rules applied by <code>PageContext.findAttribute()</code> are applied.<br><br>(RT EXPR)   |

[Back to top](#)

### **page** - Expose a specified item from the page context as a bean.

Retrieve the value of the specified item from the page context for this page, and define it as a scripting variable, and a page scope attribute accessible to the remainder of the current page.

If a problem occurs while retrieving the specified configuration object, a request time exception will be thrown.

| Attribute Name | Description  |
|----------------|--|
| id             | Specifies the name of the scripting variable (and associated page scope attribute) that will be made available with the value of the specified page context property.<br><br>(REQUIRED) (RT EXPR)  |
| property       | Name of the property from our page context to be retrieved and exposed. Must be one of <code>application</code> , <code>config</code> , <code>request</code> , <code>response</code> , or <code>session</code> .<br><br>(REQUIRED) (RT EXPR) |

[Back to top](#)

### **parameter** - Define a scripting variable based on the value(s) of the specified request parameter.

Retrieve the value of the specified request parameter (as a single value or multiple values, depending on the `multiple` attribute), and define the result as a page scope attribute of type `String` (if `multiple` is not specified) or `String[]` (if `multiple` is specified).

If no request parameter with the specified name can be located, and no default value is specified, a request time exception will be thrown.

| Attribute Name | Description   |
|----------------|---|
| id             | Specifies the name of the scripting variable (and associated page scope attribute) that will be made available with the value of the specified request parameter.<br><br>(REQUIRED) (RT EXPR)   |
| multiple       | If any arbitrary value for this attribute is specified, causes a call to <code>ServletRequest.getParameterValues()</code> and a definition of the result as a bean of type <code>String[]</code> . Otherwise, <code>ServletRequest.getParameter()</code> will be called, and a definition of the result as a bean of type <code>String</code> will be performed.<br><br>(RT EXPR) |
| name           | Specifies the name of the request parameter whose value, or values, is to be retrieved.<br><br>(REQUIRED) (RT EXPR)   |
| value          | The default parameter value to return if no parameter with the specified name was included in this request.<br><br>(RT EXPR)  |

[Back to top](#)

**resource** - Load a web application resource and make it available as a bean.

Retrieve the value of the specified web application resource, and make it available as either a `InputStream` or a `String`, depending on the value of the `input` attribute.

If a problem occurs while retrieving the specified resource, a request time exception will be thrown.

| Attribute Name | Description  |
|----------------|--|
| id             | Specifies the name of the scripting variable (and associated page scope attribute) that will be made available with the value of the specified web application resource.<br><br>(REQUIRED) (RT EXPR)   |
| input          | If any arbitrary value for this attribute is specified, the resource will be made available as an <code>InputStream</code> . If this attribute is not specified, the resource will be made available as a <code>String</code> .<br><br>(RT EXPR) |
| name           | Application-relative name (starting with a '/') of the web application resource to be loaded and made available.<br><br>(REQUIRED) (RT EXPR)   |

[Back to top](#)**size** - Define a bean containing the number of elements in a Collection or Map.

Given a reference to an array, Collection or Map, creates a new bean, of type `java.lang.Integer`, whose value is the number of elements in that collection. You can specify the collection to be counted in any one of the following ways:

- As a runtime expression specified as the value of the `collection` attribute.
- As a JSP bean specified by the `name` attribute.
- As the property, specified by the `property` attribute, of the JSP bean specified by the `bean` attribute.

| Attribute Name | Description  |
|----------------|--|
| collection     | A runtime expression that evaluates to an array, a Collection, or a Map.<br>(RT EXPR)  |
| id             | The name of a page scope JSP bean, of type <code>java.lang.Integer</code> , that will be created to contain the size of the underlying collection being counted.<br>(REQUIRED) (RT EXPR)   |
| name           | The name of the JSP bean (optionally constrained to the scope specified by the <code>scope</code> attribute) that contains the collection to be counted (if <code>property</code> is not specified), or whose property getter is called to return the collection to be counted (if <code>property</code> is specified).<br>(RT EXPR) |
| property       | The name of the property, of the bean specified by the <code>name</code> attribute, whose getter method will return the collection to be counted.<br>(RT EXPR)   |
| scope          | The bean scope within which to search for the JSP bean specified by the <code>name</code> attribute. If not specified, the available scopes are searched in ascending sequence.<br>(RT EXPR)   |

[Back to top](#)**struts** - Expose a named Struts internal configuration object as a bean.

Retrieve the value of the specified Struts internal configuration object, and define it as a scripting variable and as a page scope attribute accessible to the remainder of the current page. You must specify exactly one of the `formBean`, `forward`, and `mapping` attributes to select the configuration object to be exposed.

If a problem occurs while retrieving the specified configuration object, a request time exception will be thrown.



| Attribute Name | Description  |
|----------------|--|
| formBean       | Specifies the name of the Struts <code>ActionFormBean</code> definition object to be exposed.<br><br>(RT EXPR)   |
| forward        | Specifies the name of the global Struts <code>ActionForward</code> definition object to be exposed.<br><br>(RT EXPR)   |
| id             | Specifies the name of the scripting variable (and associated page scope attribute) that will be made available with the value of the specified Struts internal configuration object.<br><br>(REQUIRED) (RT EXPR) |
| mapping        | Specifies the matching path of the Struts <code>ActionMapping</code> definition object to be exposed.<br><br>(RT EXPR)   |

[Back to top](#)

**write** - Render the value of the specified bean property to the current `JspWriter`.

Retrieve the value of the specified bean property, and render it to the current `JspWriter` as a `String` by the ways:

- If `format` attribute exists then value will be formatted on base of format string from `format` attribute and default system locale.
- If in resources exists format string for value data type (view `format` attribute description) then value will be formatted on base of format string from resources. Resources bundle and target locale can be specified with `bundle` and `locale` attributes. If nothing specified then default resource bundle and current user locale will be used.
- If there is a `PropertyEditor` configured for the property value's class, the `getAsText()` method will be called.
- Otherwise, the usual `toString()` conversions will be applied.

If a problem occurs while retrieving the specified bean property, a request time exception will be thrown.

| Attribute Name | Description  |
|----------------|--|
| bundle         | <p>The name of the application scope bean under which the <code>MessageResources</code> object containing our messages is stored.</p> <p>[Action.MESSAGES_KEY] (RT EXPR)</p>   |
| filter         | <p>If this attribute is set to <code>true</code>, the rendered property value will be filtered for characters that are sensitive in HTML, and any such characters will be replaced by their entity equivalents.</p> <p>[true] (RT EXPR)</p>  |
| format         | <p>Specifies the format string to use to convert bean or property value to the <code>String</code>. If nothing specified, then default format string for value data type will be searched in message resources by according key.</p> <p>(RT EXPR)</p>  |
| formatKey      | <p>Specifies the key to search format string in application resources.</p> <p>(RT EXPR)</p>  |
| ignore         | <p>If this attribute is set to <code>true</code>, and the bean specified by the <code>name</code> and <code>scope</code> attributes does not exist, simply return without writing anything. If this attribute is set to <code>false</code>, a runtime exception to be thrown, consistent with the other tags in this tag library.</p> <p>[false] (RT EXPR)</p> |
| locale         | <p>The name of the session scope bean under which our currently selected <code>Locale</code> object is stored.</p> <p>[Action.LOCALE_KEY] (RT EXPR)</p>  |
| name           | <p>Specifies the attribute name of the bean whose property is accessed to retrieve the value specified by <code>property</code> (if specified). If <code>property</code> is not specified, the value of this bean itself will be rendered.</p> <p>(REQUIRED) (RT EXPR)</p>   |
| property       | <p>Specifies the name of the property to be accessed on the bean specified by <code>name</code>. This value may be a simple, indexed, or nested property reference expression. If not specified, the bean identified by <code>name</code> will itself be rendered. If the specified property returns null, no output will be rendered.</p> <p>(RT EXPR)</p>    |
| scope          | <p>Specifies the variable scope searched to retrieve the bean specified by <code>name</code>. If not specified, the default rules applied by <code>PageContext.findAttribute()</code> are applied.</p> <p>(RT EXPR)</p>  |

[Back to top](#)



[Overview](#) **Package** [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)
[PREV PACKAGE](#) [NEXT PACKAGE](#)
[FRAMES](#) [NO FRAMES](#) [All Classes](#)

# Package org.apache.struts.taglib.html

The "struts-html" tag library contains JSP custom tags useful in creating dynamic HTML user interfaces, including input forms.

See:

[Description](#)

## Class Summary

|  |  |
|--|--|
| <a href="#">BaseFieldTag</a>           | Convenience base class for the various input tags for text fields.   |
| <a href="#">BaseHandlerTag</a>         | Base class for tags that render form elements capable of including JavaScript event handlers and/or CSS Style attributes.  |
| <a href="#">BaseInputTag</a>           | Abstract base class for the various input tags.  |
| <a href="#">BaseTag</a>                | Renders an HTML element with an href attribute pointing to the absolute location of the enclosing JSP page.  |
| <a href="#">ButtonTag</a>              | Renders an HTML BUTTON tag within the Struts framework.  |
| <a href="#">CancelTag</a>              | Tag for input fields of type "cancel".   |
| <a href="#">CheckboxTag</a>            | Tag for input fields of type "checkbox".   |
| <a href="#">Constants</a>              | Manifest constants for this package.   |
| <a href="#">ErrorsTag</a>              | Custom tag that renders error messages if an appropriate request attribute has been created.   |
| <a href="#">FileTag</a>                | Custom tag for input fields of type "file".  |
| <a href="#">FormTag</a>                | Custom tag that represents an input form, associated with a bean whose properties correspond to the various fields of the form.                                    |
| <a href="#">FrameTag</a>               | Generate an HTML <frame> tag with similar capabilities as those the <html:link> tag provides for hyperlink elements.   |
| <a href="#">HiddenTag</a>              | Custom tag for input fields of type "hidden".  |
| <a href="#">HtmlTag</a>                | Renders an HTML element with appropriate language attributes if there is a current Locale available in the user's session.   |
| <a href="#">ImageTag</a>               | Tag for input fields of type "image".  |
| <a href="#">ImgTag</a>                 | Generate an IMG tag to the specified image URI.  |
| <a href="#">JavascriptValidatorTag</a> | Custom tag that generates JavaScript for client side validation based on the validation rules loaded by the ValidatorPlugIn defined in the struts-config.xml file. |
| <a href="#">LinkTag</a>                | Generate a URL-encoded hyperlink to the specified URI.   |
| <a href="#">MessagesTag</a>            | Custom tag that iterates the elements of a message collection.   |
| <a href="#">MessagesTei</a>            | Implementation of TagExtraInfo for the <b>messages</b> tag, identifying the scripting object(s) to be made visible.  |
| <a href="#">MultiboxTag</a>            | Tag for input fields of type "checkbox".   |
| <a href="#">OptionsCollectionTag</a>   | Tag for creating multiple <select> options from a collection.  |

|                             |   |
|-----------------------------|---|
| <a href="#">OptionsTag</a>  | Tag for creating multiple <select> options from a collection.   |
| <a href="#">OptionTag</a>   | Tag for select options.   |
| <a href="#">PasswordTag</a> | Custom tag for input fields of type "password".   |
| <a href="#">RadioTag</a>    | Tag for input fields of type "radio".   |
| <a href="#">ResetTag</a>    | Tag for input fields of type "reset".   |
| <a href="#">RewriteTag</a>  | Generate a URL-encoded URI as a string.   |
| <a href="#">SelectTag</a>   | Custom tag that represents an HTML select element, associated with a bean property specified by our attributes. |
| <a href="#">SubmitTag</a>   | Tag for input fields of type "submit".  |
| <a href="#">TextareaTag</a> | Custom tag for input fields of type "textarea".   |
| <a href="#">TextTag</a>     | Custom tag for input fields of type "text".   |
| <a href="#">XhtmlTag</a>    | This tag tells all other html taglib tags to render themselves in xhtml.  |

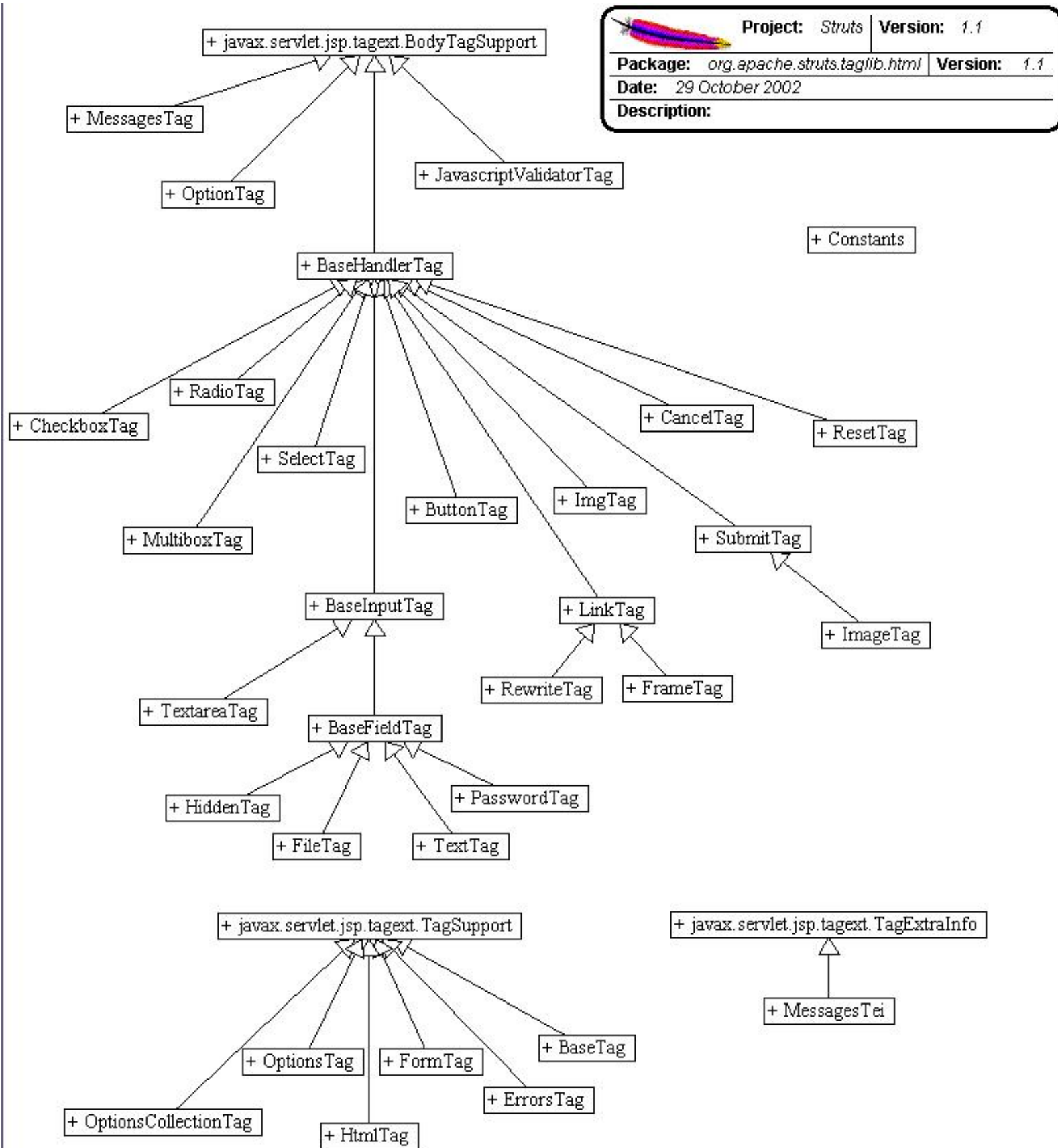
## Package org.apache.struts.taglib.html Description

The "struts-html" tag library contains JSP custom tags useful in creating dynamic HTML user interfaces, including input forms.

[\[Introduction\]](#) [\[HTML Form Tags\]](#) [\[Other HTML Tags\]](#)

### Introduction

The tags in the Struts HTML library form a bridge between a JSP view and the other components of a Web application. Since a dynamic Web application often depends on gathering data from a user, input forms play an important role in the Struts framework. Consequently, the majority of the HTML tags involve [HTML](#) forms. Other important issues addressed by the Struts-HTML tags are [messages](#) , [error messages](#) , [hyperlinking](#) and [internationalization](#) .



## HTML "form" tags

- [button](#)
- [cancel](#)

- [checkboxes](#)
- [file](#)
- [hidden](#)
- [image](#)
- [multibox](#)
- [password](#) input fields
- [radio](#) buttons
- [reset](#) buttons
- [select](#) lists with embedded
- [option](#)
- [options](#)
- [submit](#) buttons
- [text](#) input fields
- [textareas](#)

Each of these tags must be nested within a `<html:form>` tag.

## About the form tag

The Struts `form` tag outputs a standard HTML form tag, and also links the input form with a `JavaBean` subclassed from the Struts `ActionForm` object. Each field in the form should correspond to a property of the form's bean. When a field and property correspond, the bean is first used to populate the form, and then to store the user's input when the form is submitted to the controller servlet.

The name of the bean and its class can be specified as a property to the form tag, but may also be omitted. If omitted, the `ActionMappings` database (loaded from the `struts-config.xml` file) is consulted. If the current page is specified as the `input` property for an action, the name of the action is used. The `type` property for the bean is also then taken from the configuration, via a Form Bean definition.

Here's a clip from the Struts Example configuration:

```
<action-mappings>
  <!-- Process a user logon -->
  <action path="/logon"
    type="org.apache.struts.webapp.example.LogonAction"
    name="logonForm"
    scope="request"
    input="/logon.jsp">
  </action>
  < ... />
</action-mappings>
<form-beans>
  <!-- Logon form bean -->
  <form-bean name="logonForm"
    type="org.apache.struts.webapp.example.LogonForm"/>
  < ... />
</form-beans>
</pre>
```

Given this configuration, the HTML form tag for `logon.jsp` will default to using `"logonForm"` as its name property and `"org.apache.struts.webapp.example.LogonForm"` as the attribute's type. This way you can manage the namespace for your framework components from the configuration file.

If the form tag does not find the form bean in any of the scopes (page, request, session, application), it creates one using the specified type.

The Struts `ActionForm` class is equipped with standard `reset()` and `validate()` methods, that can be used by the controller to enable automatic data validation. See the [Users Guide](#) for more about Form Validation. An important aspect of validation is reporting errors to the user. This can be handled by the Struts `errors` tag, which is discussed [later in this document](#).

## Common Form Tag Attributes

The form "field" tags in the Struts-HTML tag library share a common set of tag attributes that have the same meaning, no matter what field tag they are used with. These properties also accept Runtime Expressions, meaning you can set them with a scriptlet. The common attributes fall into four categories: [Struts Common](#), [HTML Navigation](#), [Javascript](#), and [CSS](#).

### Struts Common

|          |   |
|----------|---|
| name     | The attribute name of the <code>ActionForm</code> bean whose properties are consulted when rendering the current value of this input field. If not specified, the bean associated with the form tag we are nested within is utilized. |
| property | Name of the request parameter that will be included with this submission, set to the specified value.   |
| value    | Value of the label to be used with this element. This value will also be submitted as the value of the specified request parameter. [Body of this tag (if any), or "Click"]   |

Like [Struts-Bean](#) tags, the property attribute for the Struts-HTML tags accept simple, nested, and indexed expressions. For example, this tag:

```
<html:text property="mailingAddress.street" />
```

corresponds to:

```
getMailingAddress().getStreet()
```

For more about using simple, nested, and indexed expressions with Struts, see the [Struts-Bean Developers Guide](#).

### Javascript Event Handlers

|             |  |
|-------------|--|
| onblur      | Executed when this element loses input focus.  |
| onchange    | Executed when this element loses input focus and its value has changed.                |
| onclick     | Executed when this element receives a mouse click.                                     |
| ondblclick  | Executed when this element receives a mouse - double click.                            |
| onfocus     | Executed when this element receives input focus.                                       |
| onkeydown   | Executed when this element has focus and a key is depressed.                           |
| onkeypress  | Executed when this element has focus and a key is depressed and released               |
| onkeyup     | Executed when this element has focus and a key is released                             |
| onmousedown | Executed when this element is under the mouse pointer and a mouse button is depressed. |
| onmousemove | Executed when this element is under the mouse pointer and the pointer is moved.        |



|             |   |
|-------------|---|
| onmouseout  | Executed when this element was under the mouse pointer but the pointer was moved outside the element.   |
| onmouseover | Executed when this element was not under the mouse pointer but the pointer is moved inside the element. |
| onmouseup   | Executed when this element is under the mouse pointer and a mouse button is released.                   |
|             | <b>"parent" form tag only</b>   |
| onreset     | Executed if the form is reset.  |
| onsubmit    | Executed if the form is submitted.  |

## HTML Navigation Attributes

|           |  |
|-----------|--|
| accesskey | The keyboard character used to move focus immediately to this element. |
| tabindex  | The tab order (ascending positive integers) for this element.          |

## CSS Attributes

|            |  |
|------------|--|
| style      | CSS styles to be applied to this HTML element.           |
| styleClass | CSS stylesheet class to be applied to this HTML element. |

See the [HTML Tags Reference](#) for detailed information about the available tags in this tag library, and the complete list of valid attributes for each tag.

---

## Other HTML tags

Aside from form processing, the Struts-HTML offers several other tags or tag properties to help with displaying error messages, messages, maintaining hyperlinks, and with internationalizing your application.

## Displaying Messages

Message handling is an important part of any application. These messages can be informative messages and/or error messages for the user. Struts provides a generalized method for communicating runtime messages to users, based on the same technology used to provide internationalization. Messages and error messages can both be used by the messages tag.

The messages tag [since version 1.1] has basically the same functionality as the errors tag, but it iterates through the messages so any formatting of messages can be done in the JSP page. Also the header and footer for the message tag are optional.

By default, the messages tag will iterate through all pending messages. You can also specify a property when queuing a message, and then refer to that property in the messages tag. In that case, only the message(s) for that property will be displayed. This is helpful when you would like to place the message for a field next to the actual field.

Messages are often queued in the Action. The variable info is the ActionForm corresponding to this Action. :

```

        ActionMessages messages = new ActionMessages();
        messages.add(ActionMessages.GLOBAL_MESSAGE, new
ActionMessage("userForm.insert", info.getUserName());
        messages.add("activationDate", new ActionMessage("userForm.active",
info.getSubscriptionLength()));

```

This queues two messages, one is a global message and another for the "activationDate" field. To print all the messages together, simply place the messages tag anywhere in your JSP.

```

<body bgcolor="white">
<ul>
<html:messages id="message">
    <li><bean:write name="message"/></li>
</html:messages>
</ul>

```

Or, you can place specific messages at different locations

```

<ul>
    <html:messages id="message" property="<%=
org.apache.struts.action.ActionMessages.GLOBAL_MESSAGE %>">
        <li><bean:write name="message"/></li>
    </html:messages>
</ul>
<table>
    <tr>
        <td align="left">
            <html:text property="username" size="16" maxlength="16"/>
        </td>
    </tr>
    <tr>
        <td align="left">
            <html:text property="activationDate" size="10" maxlength="10"/>
            <br>
            <html:messages id="message" property="activationDate">
                <bean:write name="message"/><br>
            </html:messages>
        </td>
    </tr>
</table>

```

By default, the actual message is retrieved from the application's standard message resource. This gives you a master list of the messages used by your application, and provides for internationalization. In the code snippet, the message corresponding to "userForm.insert" would be retrieved and displayed to the user at runtime.

```

userForm.insert={0} has successfully been inserted.
userForm.active=The account will be active for {0} months.

```

A header and footer are optional. The header will be rendered before iteration begins and the footer will be rendered after iteration is over. If a value is not assigned to the attribute, then nothing will be rendered for that attribute.

```

<html:messages id="message" header="errors.header" footer="errors.footer">
    <li><bean:write name="message"/></li>

```

```

</html:messages>
errors.header=<h3><font color="red">Validation Error</font></h3>
    You must correct the following error(s) before proceeding:<UL>
errors.footer=</ul><hr>

```

## Displaying Error Messages

Error handling is an important part of any application, and curing an error often involves getting the user's help. Struts provides a generalized method for communicating runtime messages to users, based on the same technology used to provide internationalization.

In a Web application, it is common to reuse the input page to display error messages. The Struts error message tag can be placed wherever you would like the messages to display. If no messages are pending, nothing is printed.

By default, the errors tag will print all pending messages. You can also specify a property when queuing a message, and then refer to that property in the errors tag. In that case, only the message for that property will be displayed. This is helpful when you would like to place the message for a field next to the actual field.

Error messages are often queued in the ActionForm validate method. Here's a snippet from the Struts Example application:

```

ActionErrors errors = new ActionErrors();
if ((username == null) || (username.length() < 1))
    errors.add("username", new ActionError("error.username.required"));
if ((password == null) || (password.length() < 1))
    errors.add("password",
        new ActionError("error.password.required"));

```

This queues two error messages, one for the "username" field and another for the "password" field. To print all the messages together, simply place the error tag anywhere in your JSP.

```

<body bgcolor="white">
<html:errors/>

```

Or, you can place specific error messages at different locations

```

<td align="left">
    <html:text property="username" size="16" maxlength="16"/>
    <html:errors property="username"/>
</td>
</tr><tr>
<td align="left">
    <html:text property="password" size="16" maxlength="16"/>
    <html:errors property="password"/>
</td>

```

By default, the actual error message is retrieved from the application's standard message resource. This gives you a master list of the error messages used by your application, and provides for internationalization. In the code snippet, the message corresponding to "error.username.required" would be retrieved and displayed to the user at runtime.

```

error.username.required=<li>Username is required</li>

```

Most often, error messages are displayed in a particular way. To make it easy to format your messages, you can also specify an `errors.header` and `errors.footer` string in your message resource. Typically, you might want to render the messages in a different color, or set them up to appear in an unordered list. Using the `errors.header` and `errors.footer` strings keeps the setup codes out of your JSP until a message actually prints.

```
errors.header=<h3><font color="red">Validation Error</font></h3>
    You must correct the following error(s) before proceeding:<UL>
errors.footer=</ul><hr>
```

## Maintaining Hyperlinks

- [base](#)

When implementing a Web application, it is usually a good idea to use relative references to other files in the same application. But in a dynamic application, the controller servlet often returns a completely different file than the one requested. (Often, the requested "file" doesn't actually exist, but represents an action for the controller to interpret.) Since relative references are resolved by the browser, they often won't work in a dynamic application.

To accommodate this, the Struts-HTML taglib provides a `<base>` tag that renders an HTML element with an href attribute pointing to the absolute location of its page.

As a rule, you should place the Struts-HTML base tag in the `<head>` section of any JSP that uses relative references to other files.

```
<head>
    <html:base />
</head>
```

## Session Tracking

The Java Servlet framework has built-in support for tracking a client through a session. Sessions are a useful way to compensate for the stateless nature of HTTP. Tracking a client can be done with either cookies or URL rewriting, cookies being preferred when available. Struts offers three tags that make it easy to write "session-aware" hyperlink references:

- [link](#) - Renders an HTML anchor or hyperlink.
- [rewrite](#) - Renders a request URI, but without creating the `<a>` hyperlink. This tag is useful when you want to generate a string constant for use by a JavaScript procedure.
- [img](#) - Renders an HTML `<img>` element with the image at the specified URL.

Each of these tags will automatically apply URL rewriting, to maintain session state in the absence of cookies. The content displayed for the hyperlink will be taken from the body of the tag. The base URL for the hyperlink is calculated based on properties given with the tag.

Normally, the hyperlink or URI you specify is left unchanged. If you would like to include dynamic parameters to the hyperlink or URI, you can pass those to the tags via a JavaBean. If there may be naming conflicts, you can also specify the scope that the tag searches (request, page, session, application).

To pass a single dynamic parameter, specify the parameter's name using the `paramId` attribute. You can then specify the name of the JavaBean holding the value using the `paramName` attribute, and a particular property using `paramProperty`.

So, a tag like

```
<html:link paramId="target"
    paramName="linkParams"
    paramProperty="target"
    paramScope="request"
>detail.html</html:link>
```

would correspond to something like

```
<A HREF="detail.html?<%=
```

```
org.apache.struts.taglib.html (Apache Struts API Documentation)

    request.getAttribute( "linkParams" ).getTarget( )
%> "></A>
```

To search all scopes (request, page, session, and application), omit paramScope.

The paramProperty attribute may also be omitted, so long as the named bean identifies a value that can be converted to a String.

To pass multiple dynamic parameters, you can store them in a [java.util.Map](#) , and use the name of the map for the paramName. The map must then contain one or more paramIds and their corresponding values. As the Map is processed, the keys are assumed to be the names of query parameters to be appended. The value associated with each key must be either a String or a String array representing the parameter value(s). If a String array is specified, more than one value for the same query parameter name will be created.

The HTML session tracking tags use several common attributes, that can be organized into three groups, as follows. All of these attributes are not used by all three tags (link, rewrite, and img), and so the tags using each attribute is given.

| forward, href, and page |   |
|-------------------------|---|
| forward                 | [ link rewrite ] - Logical name of a global ActionForward that contains the actual content-relative URI of the destination of this transfer. This hyperlink may be dynamically modified by the inclusion of query parameters, as described in the tag description. You must specify exactly one of the forward attribute, the href attribute, the linkName attribute, or the page attribute       |
| href                    | [ link rewrite ] - The URL to which this hyperlink will transfer control if activated. This hyperlink may be dynamically modified by the inclusion of query parameters, as described in the tag description. You must specify exactly one of the forward attribute, the href attribute, the linkName attribute, or the page attribute.  |
| page                    | [ link rewrite ] - The context-relative path, starting with a slash, of the image to be displayed by this tag. The rendered URL for this image will automatically prepend the context path of this web application (in the same manner as the page attribute on the link tag works), in addition to any necessary URL rewriting. You must specify either the page attribute or the src attribute. |

| linkName, Target , and src |  |
|----------------------------|--|
| linkName                   | [ link ] - The anchor name to be defined within this page, so that you can reference it with intra-page hyperlinks. In other words, the value specified here will render a "name" element in the generated anchor tag.   |
| target                     | [ link img ] - The window target in which the resource requested by this hyperlink will be displayed, for example in a framed presentation.  |
| src                        | [ img ] - The URL to which this image will be transferred from This image may be dynamically modified by the inclusion of query parameters, as described in the tag description. This value will be used unmodified (other than potential URL rewriting) as the value of the "src" attribute in the rendered tag. You must specify either the page attribute or the the src attribute. |

| paramId, paramName, paramProperty, paramScope |
|---|
|   |

|               |   |
|---------------|---|
| paramId       | [ link img ] - The name of the request parameter that will be dynamically added to the generated src URL. The corresponding value is defined by the paramName and (optional) paramProperty attributes, optionally scoped by the paramScope attribute  |
| paramName     | [ link img ] - The name of a JSP bean that is a String containing the value for the request parameter named by paramId (if paramProperty is not specified), or a JSP bean whose property getter is called to return a String (if paramProperty is specified). The JSP bean is constrained to the bean scope specified by the paramScope property, if it is specified. |
| paramProperty | [ link img ] - The name of a property of the bean specified by the paramName attribute, whose return value must be a String containing the value of the request parameter (named by the paramId attribute) that will be dynamically added to this hyperlink or src URL  |
| paramScope    | [ link img ] - The scope within which to search for the bean specified by the paramName attribute. If not specified, all scopes are searched.   |

| name, property, scope |  |
|-----------------------|--|
| name                  | [ link rewrite img ] - The name of a JSP bean that contains a <a href="#">Map</a> representing the query parameters (if property is not specified), or a JSP bean whose property getter is called to return a Map (if property is specified).                                    |
| property              | [ link rewrite img ] - The name of a property of the bean specified by the name attribute, whose return value must be a <a href="#">Map</a> containing the query parameters to be added to the src URL. You <b>must</b> specify the name attribute if you specify this attribute |
| scope                 | [ link rewrite img ] - The scope within which to search for the bean specified by the name attribute. If not specified, all scopes are searched.   |

See the [HTML Tags Reference](#) for the complete list of valid attributes for each tag..

## Internationalization

Internationalization is automatically supported by the HTML tags where appropriate.

[Errors](#) - By default, the text for your messages will be returned by the default message resource, which will select the appropriate language for the user. You may also specify another message resource using the `bundle` attribute.

[HTML](#) - Renders an HTML element with language attributes extracted from the user's current Locale object, if there is one.

[image](#) and [img](#) - Can optionally retrieve the value for its binary source and alt text from the message resource, so that different images and/or text can be provided for different locales. See the tag's `altKey`, `srcKey`, and `bundle` attributes in the [HTML Tags Reference](#) .

[Messages](#) - By default, the text for your messages will be returned by the default message resource, which will select the appropriate language for the user. You may also specify another message resource using the `bundle` attribute.

See the [Users Guide](#) for more on how Struts helps you internationalize your applications.

[Overview](#) **[Package](#)** [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV PACKAGE](#) [NEXT PACKAGE](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

---

Copyright © 2000-2002 - Apache Software Foundation



## User Guide

[Table of Contents](#)
[Preface](#)
[Introduction](#)
[Model](#)
[Components](#)
[View](#)
[Components](#)
[Controller](#)
[Components](#)
[Configuration](#)
[Release Notes](#)
[Installation](#)

## Developer Guides

[Bean Tags](#)
[HTML Tags](#)
[Logic Tags](#)
[Nested Tags](#)
[Template Tags](#)
[Tiles Tags](#)
[Utilities](#)
[Validator](#)

## Quick Links

[Welcome](#)
[News and Status](#)
[Resources](#)
[User and](#)
[Developer](#)
[Guides \\*](#)
[FAQs and](#)
[HowTos](#)

## Page Construction Tags

This taglib contains tags used to create struts input forms, as well as other tags generally useful in the creation of HTML-based user interfaces.

Many of the tags in this tag library will throw a `JspException` at runtime when they are utilized incorrectly (such as when you specify an invalid combination of tag attributes). JSP allows you to declare an "error page" in the `<%@ page %>` directive. If you wish to process the actual exception that caused the problem, it is passed to the error page as a request attribute under key `org.apache.struts.action.EXCEPTION`.

| Tag Name                          | Description   |
|-----------------------------------|---|
| <a href="#">base</a>              | Render an HTML <code>&lt;base&gt;</code> Element  |
| <a href="#">button</a>            | Render A Button Input Field   |
| <a href="#">cancel</a>            | Render a Cancel Button  |
| <a href="#">checkbox</a>          | Render A Checkbox Input Field   |
| <a href="#">errors</a>            | Conditionally display a set of accumulated error messages.  |
| <a href="#">file</a>              | Render A File Select Input Field  |
| <a href="#">form</a>              | Define An Input Form  |
| <a href="#">frame</a>             | Render an HTML frame element  |
| <a href="#">hidden</a>            | Render A Hidden Field   |
| <a href="#">html</a>              | Render an HTML <code>&lt;html&gt;</code> Element  |
| <a href="#">image</a>             | Render an input tag of type "image"   |
| <a href="#">img</a>               | Render an HTML <code>img</code> tag   |
| <a href="#">javascript</a>        | Render JavaScript validation based on the validation rules loaded by the <code>ValidatorPlugIn</code> . |
| <a href="#">link</a>              | Render an HTML anchor or hyperlink  |
| <a href="#">messages</a>          | Conditionally display a set of accumulated messages.  |
| <a href="#">multibox</a>          | Render A Checkbox Input Field   |
| <a href="#">option</a>            | Render A Select Option  |
| <a href="#">options</a>           | Render a Collection of Select Options   |
| <a href="#">optionsCollection</a> | Render a Collection of Select Options   |
| <a href="#">password</a>          | Render A Password Input Field   |
| <a href="#">radio</a>             | Render A Radio Button Input Field   |



|                          |                                    |
|--------------------------|------------------------------------|
| <a href="#">reset</a>    | Render A Reset Button Input Field  |
| <a href="#">rewrite</a>  | Render an URI                      |
| <a href="#">select</a>   | Render A Select Element            |
| <a href="#">submit</a>   | Render A Submit Button             |
| <a href="#">text</a>     | Render An Input Field of Type text |
| <a href="#">textarea</a> | Render A Textarea                  |
| <a href="#">xhtml</a>    | Render HTML tags as XHTML          |

### base - Render an HTML <base> Element

Renders an HTML <base> element with an href attribute pointing to the absolute location of the enclosing JSP page. This tag is valid only when nested inside an HTML <head> element.

This tag is useful because it allows you to use relative URL references in the page that are calculated based on the URL of the page itself, rather than the URL to which the most recent submit took place (which is where the browser would normally resolve relative references against).

| Attribute Name | Description   |
|----------------|---|
| server         | The server name to use instead of request.getServerName().<br>(RT EXPR) |
| target         | The window target for this base reference.<br>(RT EXPR)                 |

[Back to top](#)

### button - Render A Button Input Field

Renders an HTML <input> element of type button, populated from the specified value or the content of this tag body. This tag is only valid when nested inside a form tag body.

If a graphical button is needed (a button with an image), then the [image](#) tag is more appropriate.

| Attribute Name | Description  |
|----------------|--|
| accesskey      | The keyboard character used to move focus immediately to this element. (RT EXPR) |
| alt            | The alternate text for this element.<br>(RT EXPR)                                |
| altKey         | The message resources key of the alternate text for this element.<br>(RT EXPR)   |

|             |  |
|-------------|--|
| disabled    | Set to <code>true</code> if this input field should be disabled. (RT EXPR)   |
| indexed     | Valid only inside of <code>logic:iterate</code> tag. If <code>true</code> then name of the html tag will be rendered as "propertyName[34]". Number in brackets will be generated for every iteration and taken from ancestor <code>logic:iterate</code> tag. (RT EXPR) |
| onblur      | JavaScript event handler executed when this element loses input focus. (RT EXPR)   |
| onchange    | JavaScript event handler executed when this element loses input focus and its value has changed. (RT EXPR)   |
| onclick     | JavaScript event handler executed when this element receives a mouse click. (RT EXPR)  |
| ondblclick  | JavaScript event handler executed when this element receives a mouse double click. (RT EXPR)   |
| onfocus     | JavaScript event handler executed when this element receives input focus. (RT EXPR)  |
| onkeydown   | JavaScript event handler executed when this element has focus and a key is depressed. (RT EXPR)  |
| onkeypress  | JavaScript event handler executed when this element has focus and a key is depressed and released. (RT EXPR)   |
| onkeyup     | JavaScript event handler executed when this element has focus and a key is released. (RT EXPR)   |
| onmousedown | JavaScript event handler executed when this element is under the mouse pointer and a mouse button is depressed. (RT EXPR)  |
| onmousemove | JavaScript event handler executed when this element is under the mouse pointer and the pointer is moved. (RT EXPR)   |
| onmouseout  | JavaScript event handler executed when this element was under the mouse pointer but the pointer was moved outside the element. (RT EXPR)   |
| onmouseover | JavaScript event handler executed when this element was not under the mouse pointer but the pointer is moved inside the element. (RT EXPR)   |
| onmouseup   | JavaScript event handler executed when this element is under the mouse pointer and a mouse button is released. (RT EXPR)   |
| property    | Name of the request parameter that will be included with this submission, set to the specified value. (REQUIRED) (RT EXPR)   |
| style       | CSS styles to be applied to this HTML element. (RT EXPR)   |
| styleClass  | CSS stylesheet class to be applied to this HTML element (renders a "class" attribute). (RT EXPR)   |
| styleId     | Identifier to be assigned to this HTML element (renders an "id" attribute). (RT EXPR)  |
| tabindex    | The tab order (ascending positive integers) for this element. (RT EXPR)  |
| title       | The advisory title for this element.<br>(RT EXPR)  |
| titleKey    | The message resources key for the advisory title for this element.<br>(RT EXPR)  |

|       |  |
|-------|--|
| value | Value of the label to be placed on this button. This value will also be submitted as the value of the specified request parameter. [Body of this tag (if any), or "Click"] (RT EXPR) |
|-------|--|

[Back to top](#)

### cancel - Render a Cancel Button

Renders an HTML `<input>` element of type `submit`. This tag is only valid when nested inside a form tag body. Pressing of this submit button causes the action servlet to bypass calling the associated form bean `validate()` method. The action is called normally.

| Attribute Name | Description  |
|----------------|--|
| accesskey      | The keyboard character used to move focus immediately to this element. (RT EXPR)   |
| alt            | The alternate text for this element.<br>(RT EXPR)  |
| altKey         | The message resources key of the alternate text for this element.<br>(RT EXPR)   |
| disabled       | Set to <code>true</code> if this input field should be disabled. (RT EXPR)   |
| onblur         | JavaScript event handler executed when this element loses input focus. (RT EXPR)   |
| onchange       | JavaScript event handler executed when this element loses input focus and its value has changed. (RT EXPR)                               |
| onclick        | JavaScript event handler executed when this element receives a mouse click. (RT EXPR)  |
| ondblclick     | JavaScript event handler executed when this element receives a mouse double click. (RT EXPR)   |
| onfocus        | JavaScript event handler executed when this element receives input focus. (RT EXPR)  |
| onkeydown      | JavaScript event handler executed when this element has focus and a key is depressed. (RT EXPR)  |
| onkeypress     | JavaScript event handler executed when this element has focus and a key is depressed and released. (RT EXPR)                             |
| onkeyup        | JavaScript event handler executed when this element has focus and a key is released. (RT EXPR)   |
| onmousedown    | JavaScript event handler executed when this element is under the mouse pointer and a mouse button is depressed. (RT EXPR)                |
| onmousemove    | JavaScript event handler executed when this element is under the mouse pointer and the pointer is moved. (RT EXPR)                       |
| onmouseout     | JavaScript event handler executed when this element was under the mouse pointer but the pointer was moved outside the element. (RT EXPR) |

|             |   |
|-------------|---|
| onmouseover | JavaScript event handler executed when this element was not under the mouse pointer but the pointer is moved inside the element. (RT EXPR)  |
| onmouseup   | JavaScript event handler executed when this element is under the mouse pointer and a mouse button is released. (RT EXPR)  |
| property    | Name of the request parameter that will be included with this submission, set to the specified value. <b>WARNING</b> - If you set this attribute to a value other than the default, this will <i>NOT</i> be recognized as the cancel key by the Struts controller servlet or the <code>Action.isCancelled()</code> method. You will need to do your own cancel detection. (RT EXPR) |
| style       | CSS styles to be applied to this HTML element. (RT EXPR)  |
| styleClass  | CSS stylesheet class to be applied to this HTML element (renders a "class" attribute). (RT EXPR)  |
| styleId     | Identifier to be assigned to this HTML element (renders an "id" attribute). (RT EXPR)   |
| tabindex    | The tab order (ascending positive integers) for this element. (RT EXPR)   |
| title       | The advisory title for this element.<br>(RT EXPR)   |
| titleKey    | The message resources key for the advisory title for this element.<br>(RT EXPR)   |
| value       | Value of the label to be placed on this button. This value will also be submitted as the value of the specified request parameter. [Body of this tag (if any), or "Cancel"] (RT EXPR)   |

[Back to top](#)

## checkbox - Render A Checkbox Input Field

Renders an HTML `<input>` element of type `checkbox`, populated from the specified value or the specified property of the bean associated with our current form. This tag is only valid when nested inside a form tag body.

**NOTE:** The underlying property value associated with this field should be of type `boolean`, and any value you specify should correspond to one of the Strings that indicate a true value ("true", "yes", or "on"). If you wish to utilize a set of related String values, consider using the `multibox` tag.

**WARNING:** In order to correctly recognize unchecked checkboxes, the `ActionForm` bean associated with this form must include a statement setting the corresponding boolean property to `false` in the `reset()` method.

| Attribute Name | Description   |
|----------------|---|
| accesskey      | The keyboard character used to move focus immediately to this element. (RT EXPR)  |
| alt            | The alternate text for this element.<br>(RT EXPR)   |
| altKey         | The message resources key of the alternate text for this element.<br>(RT EXPR)  |
| disabled       | Set to <code>true</code> if this input field should be disabled. (RT EXPR)  |
| indexed        | Valid only inside of <code>logic:iterate</code> tag. If <code>true</code> then name of the html tag will be rendered as <code>"id[34].propertyName"</code> . Number in brackets will be generated for every iteration and taken from ancestor <code>logic:iterate</code> tag. (RT EXPR) |
| name           | The attribute name of the bean whose properties are consulted when rendering the current value of this input field. If not specified, the bean associated with the form tag we are nested within is utilized. (RT EXPR)   |
| onblur         | JavaScript event handler executed when this element loses input focus. (RT EXPR)  |
| onchange       | JavaScript event handler executed when this element loses input focus and its value has changed. (RT EXPR)  |
| onclick        | JavaScript event handler executed when this element receives a mouse click. (RT EXPR)   |
| ondblclick     | JavaScript event handler executed when this element receives a mouse double click. (RT EXPR)  |
| onfocus        | JavaScript event handler executed when this element receives input focus. (RT EXPR)   |
| onkeydown      | JavaScript event handler executed when this element has focus and a key is depressed. (RT EXPR)   |
| onkeypress     | JavaScript event handler executed when this element has focus and a key is depressed and released. (RT EXPR)  |
| onkeyup        | JavaScript event handler executed when this element has focus and a key is released. (RT EXPR)  |
| onmousedown    | JavaScript event handler executed when this element is under the mouse pointer and a mouse button is depressed. (RT EXPR)   |
| onmousemove    | JavaScript event handler executed when this element is under the mouse pointer and the pointer is moved. (RT EXPR)  |
| onmouseout     | JavaScript event handler executed when this element was under the mouse pointer but the pointer was moved outside the element. (RT EXPR)  |
| onmouseover    | JavaScript event handler executed when this element was not under the mouse pointer but the pointer is moved inside the element. (RT EXPR)  |
| onmouseup      | JavaScript event handler executed when this element is under the mouse pointer and a mouse button is released. (RT EXPR)  |
| property       | Name of the request parameter that will be included with this submission, set to the specified value. (REQUIRED) (RT EXPR)  |
| style          | CSS styles to be applied to this HTML element. (RT EXPR)  |

|            |  |
|------------|--|
| styleClass | CSS stylesheet class to be applied to this HTML element (renders a "class" attribute). (RT EXPR)   |
| styleId    | Identifier to be assigned to this HTML element (renders an "id" attribute). (RT EXPR)  |
| tabindex   | The tab order (ascending positive integers) for this element. (RT EXPR)  |
| title      | The advisory title for this element.<br>(RT EXPR)  |
| titleKey   | The message resources key for the advisory title for this element.<br>(RT EXPR)  |
| value      | The value to be transmitted if this checkbox is checked when the form is submitted. If not specified, the value "on" will be returned. (RT EXPR) |

[Back to top](#)

### errors - Conditionally display a set of accumulated error messages.

Displays a set of error messages prepared by a business logic component and stored as an `ActionErrors` object, a `String`, or a `String` array in request scope. If such a bean is not found, nothing will be rendered.

In order to use this tag successfully, you must have defined an application scope `MessageResources` bean under the default attribute name, with optional definitions of the following message keys:

- **errors.header** - Text that will be rendered before the error messages list. Typically, this message text will end with `<ul>` to start the error messages list.
- **errors.footer** - Text that will be rendered after the error messages list. Typically, this message text will begin with `</ul>` to end the error messages list.
- **errors.prefix** - Text that will be rendered before each individual error in the list.
- **errors.suffix** - Text that will be rendered after each individual error in the list.

| Attribute Name | Description   |
|----------------|---|
| bundle         | The servlet context attribute key for the <code>MessageResources</code> instance to use. If not specified, defaults to the application resources configured for our action servlet. (RT EXPR)   |
| locale         | The session attribute key for the <code>Locale</code> used to select messages to be displayed. If not specified, defaults to the Struts standard value. (RT EXPR)                               |
| name           | Name of the request scope bean under which our error messages have been stored. If not present, the name specified by the <code>Action.ERROR_KEY</code> constant string will be used. (RT EXPR) |
| property       | Name of the property for which error messages should be displayed. If not specified, all error messages (regardless of property) are displayed. (RT EXPR)                                       |

[Back to top](#)**file** - Render A File Select Input Field

Renders an HTML `<input>` element of type `file`, defaulting to the specified value or the specified property of the bean associated with our current form. This tag is only valid when nested inside a form tag body.

As with the corresponding HTML `<input>` element, the enclosing form element must specify "POST" for the method attribute, and "multipart/form-data" for the `enctype` attribute. For example:

```
<html:form method="POST" enctype="multipart/form-data">
  <html:file property="theFile" />
</html:form>
```

| Attribute Name | Description  |
|----------------|--|
| accept         | Comma-delimited set of content types that the server you submit to knows how to process. This list can be used by the client browser to limit the set of file options that is made available for selection. If not specified, no content type list will be sent. (RT EXPR) |
| accesskey      | The keyboard character used to move focus immediately to this element. (RT EXPR)   |
| alt            | The alternate text for this element.<br>(RT EXPR)  |
| altKey         | The message resources key of the alternate text for this element.<br>(RT EXPR)   |
| disabled       | Set to <code>true</code> if this input field should be disabled. (RT EXPR)   |
| indexed        | Valid only inside of <code>logic:iterate</code> tag. If <code>true</code> then name of the html tag will be rendered as "id[34].propertyName". Number in brackets will be generated for every iteration and taken from ancestor <code>logic:iterate</code> tag. (RT EXPR)  |
| maxlength      | Maximum number of input characters to accept. [No limit] (RT EXPR)   |
| name           | The attribute name of the bean whose properties are consulted when rendering the current value of this input field. If not specified, the bean associated with the form tag we are nested within is utilized. (RT EXPR)  |
| onblur         | JavaScript event handler executed when this element loses input focus. (RT EXPR)   |
| onchange       | JavaScript event handler executed when this element loses input focus and its value has changed. (RT EXPR)   |
| onclick        | JavaScript event handler executed when this element receives a mouse click. (RT EXPR)  |
| ondblclick     | JavaScript event handler executed when this element receives a mouse double click. (RT EXPR)   |
| onfocus        | JavaScript event handler executed when this element receives input focus. (RT EXPR)  |

|             |   |
|-------------|---|
| onkeydown   | JavaScript event handler executed when this element has focus and a key is depressed. (RT EXPR)   |
| onkeypress  | JavaScript event handler executed when this element has focus and a key is depressed and released. (RT EXPR)  |
| onkeyup     | JavaScript event handler executed when this element has focus and a key is released. (RT EXPR)  |
| onmousedown | JavaScript event handler executed when this element is under the mouse pointer and a mouse button is depressed. (RT EXPR)   |
| onmousemove | JavaScript event handler executed when this element is under the mouse pointer and the pointer is moved. (RT EXPR)  |
| onmouseout  | JavaScript event handler executed when this element was under the mouse pointer but the pointer was moved outside the element. (RT EXPR)  |
| onmouseover | JavaScript event handler executed when this element was not under the mouse pointer but the pointer is moved inside the element. (RT EXPR)  |
| onmouseup   | JavaScript event handler executed when this element is under the mouse pointer and a mouse button is released. (RT EXPR)  |
| property    | Name of the request parameter that will be included with this submission, set to the specified value. (REQUIRED) (RT EXPR)  |
| size        | Size of the file selection box to be displayed. (RT EXPR)   |
| style       | CSS styles to be applied to this HTML element. (RT EXPR)  |
| styleClass  | CSS stylesheet class to be applied to this HTML element (renders a "class" attribute). (RT EXPR)  |
| styleId     | Identifier to be assigned to this HTML element (renders an "id" attribute). (RT EXPR)   |
| tabindex    | The tab order (ascending positive integers) for this element. (RT EXPR)   |
| title       | The advisory title for this element.<br>(RT EXPR)   |
| titleKey    | The message resources key for the advisory title for this element.<br>(RT EXPR)   |
| value       | Value of the label to be placed on this button. This value will also be submitted as the value of the specified request parameter. [Body of this tag (if any), or "Cancel"] (RT EXPR) |

[Back to top](#)

## form - Define An Input Form



Renders an HTML `<form>` element whose contents are described by the body content of this tag. The form implicitly interacts with the specified request scope or session scope bean to populate the input fields with the current property values from the bean.

The associated form bean is determined in one of two ways:

- If the `name` and `type` attributes are not specified, then the form bean will be located, and created if necessary, based on the form bean specification for the associated `ActionMapping`.
- If the `name` and `type` attributes are specified, then the form bean will be located, and created if necessary, using the specified values for `name`, `type` and `scope`.

**NOTE:** The use of the `name`, `type` and `scope` attributes is deprecated. The preferred usage is to allow the appropriate values to be determined automatically from the corresponding `ActionMapping`.

| Attribute Name | Description  |
|----------------|--|
| action         | <p>The URL to which this form will be submitted. This value is also used to select the <code>ActionMapping</code> we are assumed to be processing, from which we can identify the appropriate form bean and scope.</p> <p>If you are using extension mapping for selecting the controller servlet, this value should be equal to the <code>path</code> attribute of the corresponding <code>&lt;action&gt;</code> element, optionally followed by the correct extension suffix.</p> <p>If you are using path mapping to select the controller servlet, this value should be exactly equal to the <code>path</code> attribute of the corresponding <code>&lt;action&gt;</code> element.</p> <p>(REQUIRED) (RT EXPR)</p> |
| enctype        | <p>The content encoding to be used to submit this form, if the method is POST. This must be set to "multipart/form-data" if you are using the file tag to enable file upload. If not specified, the browser default (normally "application/x-www-form-urlencoded") is used. (RT EXPR)</p>  |
| focus          | <p>The field name (among the fields on this form) to which initial focus will be assigned with a JavaScript function. If not specified, no special JavaScript for this purpose will be rendered. (RT EXPR)</p>   |
| focusIndex     | <p>Since: Struts 1.1</p> <p>If the focus field is a field array, such as a radio button group, you can specify the index in the array to receive focus. (RT EXPR)</p>  |
| method         | <p>The HTTP method that will be used to submit this request (GET, POST). [POST] (RT EXPR)</p>  |

|            |   |
|------------|---|
| name       | <p><b>DEPRECATED:</b> The bean name will be determined from the corresponding <code>ActionMapping</code>.</p> <p>Name of the request scope or session scope bean (as defined by the <code>scope</code> attribute) whose properties will be used to populate the input field values. If no such bean is found, a new bean will be created and added to the appropriate scope, using the Java class name specified by the <code>type</code> attribute.</p> <p>If this attribute is not specified, the name of the bean will be calculated by using the value of the <code>action</code> attribute to look up the corresponding <code>ActionMapping</code> element, from which the specified form bean name will be selected.</p> <p>(RT EXPR)</p> |
| onreset    | JavaScript event handler executed if the form is reset. (RT EXPR)   |
| onsubmit   | JavaScript event handler executed if the form is submitted. (RT EXPR)   |
| scope      | <p><b>DEPRECATED:</b> The bean scope will be determined from the corresponding <code>ActionMapping</code>.</p> <p>Scope within which the form bean associated with this input form will be accessed or created (must be either <code>request</code> or <code>session</code>).</p> <p>If this attribute is not specified, the scope of the bean will be calculated by using the value of the <code>action</code> attribute to look up the corresponding <code>ActionMapping</code> element, from which the specified form bean scope will be selected.</p> <p>(RT EXPR)</p>  |
| style      | CSS styles to be applied to this HTML element. (RT EXPR)  |
| styleClass | CSS stylesheet class to be applied to this HTML element (renders a "class" attribute). (RT EXPR)  |
| styleId    | Identifier to be assigned to this HTML element (renders an "id" attribute). (RT EXPR)   |
| target     | Window target to which this form is submitted, such as for use in framed presentations. (RT EXPR)   |
| type       | <p><b>DEPRECATED:</b> The bean type will be determined from the corresponding <code>ActionMapping</code>.</p> <p>Fully qualified Java class name of the form bean to be created, if no such bean is found in the specified scope.</p> <p>If this attribute is not specified, the type of the bean will be calculated by using the value of the <code>action</code> attribute to look up the corresponding <code>ActionMapping</code> element, from which the specified form bean type will be selected.</p> <p>(RT EXPR)</p>  |

[Back to top](#)

## frame - Render an HTML frame element

Renders an HTML `<frame>` element with processing for the `src` attribute that is identical to that performed by the `<html:link>` tag for the `href` attribute. URL rewriting will be applied automatically, to maintain session state in the absence of cookies.

The base URL for this frame is calculated based on which of the following attributes you specify (you must specify exactly one of them):

- *forward* - Use the value of this attribute as the name of a global `ActionForward` to be looked up, and use the application-relative or context-relative URI found there.
- *href* - Use the value of this attribute unchanged.
- *page* - Use the value of this attribute as a application-relative URI, and generate a server-relative URI by including the context path and application prefix.

Normally, the hyperlink you specify with one of the attributes described in the previous paragraph will be left unchanged (other than URL rewriting if necessary). However, there are two ways you can append one or more dynamically defined query parameters to the hyperlink -- specify a single parameter with the `paramId` attribute (and its associated attributes to select the value), or specify the name (and optional `property`) attributes to select a `java.util.Map` bean that contains one or more parameter ids and corresponding values.

To specify a single parameter, use the `paramId` attribute to define the name of the request parameter to be submitted. To specify the corresponding value, use one of the following approaches:

- *Specify only the paramName attribute* - The named JSP bean (optionally scoped by the value of the `paramScope` attribute) must identify a value that can be converted to a `String`.
- *Specify both the paramName and paramProperty attributes* - The specified property getter method will be called on the JSP bean identified by the `paramName` (and optional `paramScope`) attributes, in order to select a value that can be converted to a `String`.

If you prefer to specify a `java.util.Map` that contains all of the request parameters to be added to the hyperlink, use one of the following techniques:

- *Specify only the name attribute* - The named JSP bean (optionally scoped by the value of the `scope` attribute) must identify a `java.util.Map` containing the parameters.
- *Specify both name and property attributes* - The specified property getter method will be called on the bean identified by the `name` (and optional `scope`) attributes, in order to return the `java.util.Map` containing the parameters.

As the `Map` is processed, the keys are assumed to be the names of query parameters to be appended to the hyperlink. The value associated with each key must be either a `String` or a `String` array representing the parameter value(s), or an object whose `toString()` method will be called. If a `String` array is specified, more than one value for the same query parameter name will be created.

Additionally, you can request that the current transaction control token, if any, be included in the generated hyperlink by setting the `transaction` attribute to `true`. You can also request that an anchor ("`#xxx`") be added to the end of the URL that is created by any of the above mechanisms, by using the `anchor` attribute.

| Attribute Name | Description   |
|----------------|---|
| action         | <p>Logical name of a global <code>Action</code> that contains the actual content-relative URI of the destination of this transfer. This hyperlink may be dynamically modified by the inclusion of query parameters, as described in the tag description. You <b>must</b> specify exactly one of the <code>action</code> attribute, the <code>forward</code> attribute, the <code>href</code> attribute, or the <code>page</code> attribute.</p> <p>(RT EXPR)</p>        |
| anchor         | <p>Optional anchor tag ("<code>#xxx</code>") to be added to the generated hyperlink. Specify this value <b>without</b> any "<code>#</code>" character.</p> <p>(RT EXPR)</p>   |
| forward        | <p>Logical name of a global <code>ActionForward</code> that contains the actual content-relative URI of the destination of this transfer. This hyperlink may be dynamically modified by the inclusion of query parameters, as described in the tag description. You <b>must</b> specify exactly one of the <code>action</code> attribute, the <code>forward</code> attribute, the <code>href</code> attribute, or the <code>page</code> attribute.</p> <p>(RT EXPR)</p> |
| frameborder    | <p>Should a frame border be generated around this frame (1) or not (0)?</p> <p>(RT EXPR)</p>  |
| frameName      | <p>Value for the name attribute of the rendered <code>&lt;frame&gt;</code> element.</p> <p>(RT EXPR)</p>  |
| href           | <p>The URL to which this hyperlink will transfer control if activated. This hyperlink may be dynamically modified by the inclusion of query parameters, as described in the tag description. You <b>must</b> specify exactly one of the <code>action</code> attribute, the <code>forward</code> attribute, the <code>href</code> attribute, or the <code>page</code> attribute.</p> <p>(RT EXPR)</p>  |
| longdesc       | <p>URI of a long description of the frame. This description should supplement the short description provided by the <code>title</code> attribute, and may be particularly useful for non-visual user agents.</p> <p>(RT EXPR)</p>   |
| marginheight   | <p>The amount of space (in pixels) to be left between the frame's contents and its top and bottom margins.</p> <p>(RT EXPR)</p>   |
| marginwidth    | <p>The amount of space (in pixels) to be left between the frame's contents and its left and right margins.</p> <p>(RT EXPR)</p>   |

|               |   |
|---------------|---|
| name          | <p>The name of a JSP bean that contains a Map representing the query parameters (if <code>property</code> is not specified), or a JSP bean whose property getter is called to return a Map (if <code>property</code> is specified).</p> <p>(RT EXPR)</p>  |
| noresize      | <p>Should users be disallowed from resizing the frame? (true, false).</p> <p>(RT EXPR)</p>  |
| page          | <p>The application-relative path (beginning with a "/" character) to which this hyperlink will transfer control if activated. This hyperlink may be dynamically modified by the inclusion of query parameters, as described in the tag description. You <b>must</b> specify exactly one of the <code>action</code> attribute, the <code>forward</code> attribute, the <code>href</code> attribute, or the <code>page</code> attribute.</p> <p>(RT EXPR)</p> |
| paramId       | <p>The name of the request parameter that will be dynamically added to the generated hyperlink. The corresponding value is defined by the <code>paramName</code> and (optional) <code>paramProperty</code> attributes, optionally scoped by the <code>paramScope</code> attribute</p> <p>(RT EXPR)</p>  |
| paramName     | <p>The name of a JSP bean that is a String containing the value for the request parameter named by <code>paramId</code> (if <code>paramProperty</code> is not specified), or a JSP bean whose property getter is called to return a String (if <code>paramProperty</code> is specified). The JSP bean is constrained to the bean scope specified by the <code>paramScope</code> property, if it is specified.</p> <p>(RT EXPR)</p>                          |
| paramProperty | <p>The name of a property of the bean specified by the <code>paramName</code> attribute, whose return value must be a String containing the value of the request parameter (named by the <code>paramId</code> attribute) that will be dynamically added to this hyperlink.</p> <p>(RT EXPR)</p>   |
| paramScope    | <p>The scope within which to search for the bean specified by the <code>paramName</code> attribute. If not specified, all scopes are searched.</p> <p>(RT EXPR)</p>   |
| property      | <p>The name of a property of the bean specified by the <code>name</code> attribute, whose return value must be a <code>java.util.Map</code> containing the query parameters to be added to the hyperlink. You <b>must</b> specify the <code>name</code> attribute if you specify this attribute.</p> <p>(RT EXPR)</p>   |
| scope         | <p>The scope within which to search for the bean specified by the <code>name</code> attribute. If not specified, all scopes are searched.</p> <p>(RT EXPR)</p>  |

|             |   |
|-------------|---|
| scrolling   | Should scroll bars be created unconditionally (yes), never (no), or only when needed (auto)?<br><br>(RT EXPR)   |
| style       | CSS styles to be applied to this element.<br><br>(RT EXPR)  |
| styleClass  | CSS stylesheet class to be applied to this HTML element (renders a "class" attribute). (RT EXPR)  |
| styleId     | Identifier to be assigned to this HTML element (renders an "id" attribute). (RT EXPR)   |
| title       | The advisory title for this element.<br><br>(RT EXPR)   |
| titleKey    | The message resources key for the advisory title for this element.<br><br>(RT EXPR)   |
| transaction | If set to <code>true</code> , any current transaction control token will be included in the generated hyperlink, so that it will pass an <code>isTokenValid()</code> test in the receiving Action.<br><br>(RT EXPR) |

[Back to top](#)

### hidden - Render A Hidden Field

Renders an HTML `<input>` element of type `hidden`, populated from the specified value or the specified property of the bean associated with our current form. This tag is only valid when nested inside a form tag body.

| Attribute Name | Description   |
|----------------|---|
| accesskey      | The keyboard character used to move focus immediately to this element. (RT EXPR)  |
| alt            | The alternate text for this element.<br><br>(RT EXPR)   |
| altKey         | The message resources key of the alternate text for this element.<br><br>(RT EXPR)  |
| indexed        | Valid only inside of <code>logic:iterate</code> tag. If <code>true</code> then name of the html tag will be rendered as <code>"id[34].propertyName"</code> . Number in brackets will be generated for every iteration and taken from ancestor <code>logic:iterate</code> tag. (RT EXPR) |
| name           | The attribute name of the bean whose properties are consulted when rendering the current value of this input field. If not specified, the bean associated with the form tag we are nested within is utilized. (RT EXPR)   |

|             |   |
|-------------|---|
| onblur      | JavaScript event handler executed when this element loses input focus. (RT EXPR)  |
| onchange    | JavaScript event handler executed when this element loses input focus and its value has changed. (RT EXPR)  |
| onclick     | JavaScript event handler executed when this element receives a mouse click. (RT EXPR)   |
| ondblclick  | JavaScript event handler executed when this element receives a mouse double click. (RT EXPR)  |
| onfocus     | JavaScript event handler executed when this element receives input focus. (RT EXPR)   |
| onkeydown   | JavaScript event handler executed when this element has focus and a key is depressed. (RT EXPR)   |
| onkeypress  | JavaScript event handler executed when this element has focus and a key is depressed and released. (RT EXPR)  |
| onkeyup     | JavaScript event handler executed when this element has focus and a key is released. (RT EXPR)  |
| onmousedown | JavaScript event handler executed when this element is under the mouse pointer and a mouse button is depressed. (RT EXPR)   |
| onmousemove | JavaScript event handler executed when this element is under the mouse pointer and the pointer is moved. (RT EXPR)  |
| onmouseout  | JavaScript event handler executed when this element was under the mouse pointer but the pointer was moved outside the element. (RT EXPR)  |
| onmouseover | JavaScript event handler executed when this element was not under the mouse pointer but the pointer is moved inside the element. (RT EXPR)  |
| onmouseup   | JavaScript event handler executed when this element is under the mouse pointer and a mouse button is released. (RT EXPR)  |
| property    | Name of this input field, and the name of the corresponding bean property if value is not specified. The corresponding bean property (if any) must be of type String. (REQUIRED) (RT EXPR)                |
| style       | CSS styles to be applied to this HTML element. (RT EXPR)  |
| styleClass  | CSS stylesheet class to be applied to this HTML element (renders a "class" attribute). (RT EXPR)  |
| styleId     | Identifier to be assigned to this HTML element (renders an "id" attribute). (RT EXPR)   |
| title       | The advisory title for this element.<br>(RT EXPR)   |
| titleKey    | The message resources key for the advisory title for this element.<br>(RT EXPR)   |
| value       | Value to which this field should be initialized. [Use the corresponding bean property value] (RT EXPR)  |
| write       | Should the value of this field also be rendered to the response page to make it visible, in addition to creating an HTML type="hidden" element? By default, only the hidden element is created. (RT EXPR) |

[Back to top](#)**html** - Render an HTML `<html>` Element

Renders an HTML `<html>` element with language attributes extracted from the user's current Locale object, if there is one.

| Attribute Name | Description   |
|----------------|---|
| locale         | Set to <code>true</code> in order to record a Locale based on the current request's <code>Accept-Language</code> header (if any) if none has currently been set.<br><br>(RT EXPR)   |
| xhtml          | Set to <code>true</code> in order to render an <code>xml:lang</code> attribute on the generated <code>html</code> element. This also causes all other <code>html</code> tags to render as <code>xhtml</code> .<br><br>(RT EXPR) |

[Back to top](#)**image** - Render an input tag of type "image"

Renders an HTML `<input>` tag of type "image". The base URL for this image is calculated directly based on the value specified in the `src` or `page` attributes, or indirectly by looking up a message resource string based on the `srcKey` or `pageKey` attributes. You **must** specify exactly one of these attributes.

If you would like to obtain the coordinates of the mouse click that submitted this request, see the information below on the `property` attribute.

This tag is only valid when nested inside a form tag body.

| Attribute Name | Description   |
|----------------|---|
| accesskey      | The keyboard character used to move focus immediately to this element.<br><br>(RT EXPR)   |
| align          | The alignment option for this image.<br><br>The alignment option for this image. The <code>align</code> attribute is deprecated in HTML 4.x. The suggested alternative is to use CSS. Please see <a href="http://www.w3.org/TR/REC-html40/struct/objects.html#h-13.7.4">http://www.w3.org/TR/REC-html40/struct/objects.html#h-13.7.4</a> for more details.<br><br>(RT EXPR) |
| alt            | The alternate text for this image.<br><br>(RT EXPR)   |



|             |  |
|-------------|--|
| altKey      | The message resources key of the alternate text for this image.<br>(RT EXPR)   |
| border      | The width (in pixels) of the border around this image.<br>(RT EXPR)  |
| bundle      | The servlet context attribute key for the MessageResources instance to use. If not specified, defaults to the application resources configured for our action servlet.<br>(RT EXPR)  |
| disabled    | Set to <code>true</code> if this input field should be disabled. (RT EXPR)   |
| indexed     | Valid only inside of <code>logic:iterate</code> tag. If <code>true</code> then name of the html tag will be rendered as "propertyName[34]". Number in brackets will be generated for every iteration and taken from ancestor <code>logic:iterate</code> tag. (RT EXPR) |
| locale      | The session attribute key for the Locale used to select internationalized messages. If not specified, defaults to the Struts standard value.<br>(RT EXPR)  |
| onblur      | JavaScript event handler executed when this element loses input focus. (RT EXPR)   |
| onchange    | JavaScript event handler executed when this element loses input focus and its value has changed. (RT EXPR)   |
| onclick     | JavaScript event handler executed when this element receives a mouse click. (RT EXPR)  |
| ondblclick  | JavaScript event handler executed when this element receives a mouse double click. (RT EXPR)   |
| onfocus     | JavaScript event handler executed when this element receives input focus. (RT EXPR)  |
| onkeydown   | JavaScript event handler executed when this element has focus and a key is depressed. (RT EXPR)  |
| onkeypress  | JavaScript event handler executed when this element has focus and a key is depressed and released. (RT EXPR)   |
| onkeyup     | JavaScript event handler executed when this element has focus and a key is released. (RT EXPR)   |
| onmousedown | JavaScript event handler executed when this element is under the mouse pointer and a mouse button is depressed. (RT EXPR)  |
| onmousemove | JavaScript event handler executed when this element is under the mouse pointer and the pointer is moved. (RT EXPR)   |
| onmouseout  | JavaScript event handler executed when this element was under the mouse pointer but the pointer was moved outside the element. (RT EXPR)   |
| onmouseover | JavaScript event handler executed when this element was not under the mouse pointer but the pointer is moved inside the element. (RT EXPR)   |
| onmouseup   | JavaScript event handler executed when this element is under the mouse pointer and a mouse button is released. (RT EXPR)   |

|            |   |
|------------|---|
| page       | The application-relative path of the image for this input tag.<br>(RT EXPR)   |
| pageKey    | The key of the message resources string specifying the application-relative path of the image for this input tag.<br>(RT EXPR)  |
| property   | The property name of this image tag. The parameter names for the request will appear as "property.x" and "property.y", the x and y representing the coordinates of the mouse click for the image. A way of retrieving these values through a form bean is to define getX(), getY(), setX(), and setY() methods, and specify your property as a blank string (property="").<br>(RT EXPR) |
| src        | The source URL of the image for this input tag.<br>(RT EXPR)  |
| srcKey     | The key of the message resources string specifying the source URL of the image for this input tag.<br>(RT EXPR)   |
| style      | CSS styles to be applied to this HTML element. (RT EXPR)  |
| styleClass | CSS stylesheet class to be applied to this HTML element (renders a "class" attribute). (RT EXPR)  |
| styleId    | Identifier to be assigned to this HTML element (renders an "id" attribute). (RT EXPR)   |
| tabindex   | The tab order (ascending positive integers) for this element. (RT EXPR)   |
| title      | The advisory title for this element.<br>(RT EXPR)   |
| titleKey   | The message resources key for the advisory title for this element.<br>(RT EXPR)   |
| value      | The value that will be submitted if this image button is pressed.<br>(RT EXPR)  |

[Back to top](#)

## img - Render an HTML img tag

Renders an HTML `<img>` element with the image at the specified URL. Like the link tag, URL rewriting will be applied automatically to the value specified in `src` or `page`, to maintain session state in the absence of cookies. This will allow dynamic generation of an image where the content displayed for this image will be taken from the attributes of this tag.

The base URL for this image is calculated directly based on the value specified in `src` or `page`, or indirectly by looking up a message resource string based on the `srcKey` or `pageKey` attributes. You **must** specify exactly one of these attributes.

Normally, the `src` or `page` that you specify will be left unchanged (other than URL rewriting if necessary). However, there are two ways you can append one or more dynamically defined query parameters to the `src` URL -- specify a single parameter with the `paramId` attribute (at its associated attributes to select the value), or specify the name (and optional `property`) attributes to select a `java.util.Map` bean that contains one or more parameter ids and corresponding values.

To specify a single parameter, use the `paramId` attribute to define the name of the request parameter to be submitted. To specify the corresponding value, use one of the following approaches:

- *Specify only the `paramName` attribute* - The named JSP bean (optionally scoped by the value of the `paramScope` attribute) must identify a value that can be converted to a `String`.
- *Specify both the `paramName` and `paramProperty` attributes* - The specified property getter will be called on the JSP bean identified by the `paramName` (and optional `paramScope`) attributes, in order to select a value that can be converted to a `String`.

If you prefer to specify a `java.util.Map` that contains all of the request parameters to be added to the hyperlink, use one of the following techniques:

- *Specify only the `name` attribute* - The named JSP bean (optionally scoped by the value of the `scope` attribute) must identify a `java.util.Map` containing the parameters.
- *Specify both `name` and `property` attributes* - The specified property getter method will be called on the bean identified by the `name` (and optional `scope`) attributes, in order to return the `java.util.Map` containing the parameters.

As the Map is processed, the keys are assumed to be the names of query parameters to be appended to the `src` URL. The value associated with each key must be either a `String` or a `String` array representing the parameter value(s), or an object whose `toString()` method will be called. If a `String` array is specified, more than one value for the same query parameter name will be created.

You can specify the alternate text for this image (which most browsers display as pop-up text block when the user hovers the mouse over this image) either directly, through the `alt` attribute, or indirectly from a message resources bundle, using the `bundle` and `altKey` attributes.

| Attribute Name | Description   |
|----------------|---|
| align          | <p>Where the image is aligned to. Can be one of the following attributes:</p> <ul style="list-style-type: none"> <li>• left - left justify, wrapping text on right</li> <li>• right -right justify, wrapping test on left</li> <li>• top - aligns the image with the top of the text on the same row</li> <li>• middle - aligns the image's vertical center with the text base line</li> <li>• bottom - aligns the image with the bottom of the text's base line</li> <li>• texttop - aligns the image's top with that of the text font on the same line</li> <li>• absmiddle - aligns the image's vertical center with the absolute center of the text</li> <li>• absbottom - aligns the image with the absolute bottom of the text font on the same row</li> </ul> <p>(RT EXPR)</p> |
| alt            | <p>And alternative text to be displayed in browsers that don't support graphics. Also used often as type of context help over images.</p> <p>(RT EXPR)</p>  |
| altKey         | <p>The message resources key of the alternate text for this element.</p> <p>(RT EXPR)</p>   |
| border         | <p>The width of the border surrounding the image.</p> <p>(RT EXPR)</p>  |
| bundle         | <p>The servlet context attribute key for the MessageResources instance to use. If not specified, defaults to the application resources configured for our action servlet.</p> <p>(RT EXPR)</p>  |
| height         | <p>The height of the image being displayed. This parameter is very nice to specify (along with <code>width</code>) to help the browser render the page faster.</p> <p>(RT EXPR)</p>   |
| hspace         | <p>The amount of horizontal spacing between the icon and the text. The text may be in the same paragraph, or be wrapped around the image.</p> <p>(RT EXPR)</p>  |
| imageName      | <p>The scriptable name to be defined within this page, so that you can reference it with intra-page scripts. In other words, the value specified here will render a "name" element in the generated image tag.</p> <p>(RT EXPR)</p>   |
| ismap          | <p>The name of the server-side map that this image belongs to.</p> <p>(RT EXPR)</p>   |

|             |  |
|-------------|--|
| locale      | The name of the request or session Locale attribute used to look up internationalized messages.<br><br>(RT EXPR)   |
| lowsrc      | <b>DEPRECATED</b> - This attribute is not defined in the HTML 4.01 spec and will be removed in a future version of Struts. An image for people with low resolution graphics cards.<br><br>(RT EXPR)  |
| name        | The name of a JSP bean that contains a Map representing the query parameters (if <code>property</code> is not specified), or a JSP bean whose property getter is called to return a Map (if <code>property</code> is specified).<br><br>(RT EXPR)  |
| onclick     | JavaScript event handler executed when this element receives a mouse click.<br>(RT EXPR)   |
| ondblclick  | JavaScript event handler executed when this element receives a mouse double click. (RT EXPR)   |
| onkeydown   | JavaScript event handler that is executed when this element receives a key down event.<br><br>(RT EXPR)  |
| onkeypress  | JavaScript event handler that is executed when this element receives a key press event.<br><br>(RT EXPR)   |
| onkeyup     | JavaScript event handler that is executed when this element receives a key up event.<br><br>(RT EXPR)  |
| onmousedown | JavaScript event handler executed when this element is under the mouse pointer and a mouse button is depressed. (RT EXPR)  |
| onmousemove | JavaScript event handler executed when this element is under the mouse pointer and the pointer is moved. (RT EXPR)   |
| onmouseout  | JavaScript event handler executed when this element was under the mouse pointer but the pointer was moved outside the element. (RT EXPR)   |
| onmouseover | JavaScript event handler executed when this element was not under the mouse pointer but the pointer is moved inside the element. (RT EXPR)   |
| onmouseup   | JavaScript event handler executed when this element is under the mouse pointer and a mouse button is released. (RT EXPR)   |
| page        | The application-relative path, starting with a slash, of the image to be displayed by this tag. The rendered URL for this image will automatically prepend the context path of this web application (in the same manner as the <code>page</code> attribute on the <code>link</code> tag works), in addition to any necessary URL rewriting. You <b>must</b> specify either the <code>page</code> attribute or the <code>src</code> attribute.<br><br>(RT EXPR) |

|               |  |
|---------------|--|
| pageKey       | <p>The message key, in the message resources bundle named by the <code>bundle</code> attribute, of the String to be used as the application-relative path for this image.</p> <p>(RT EXPR)</p>   |
| paramId       | <p>The name of the request parameter that will be dynamically added to the generated src URL. The corresponding value is defined by the <code>paramName</code> and (optional) <code>paramProperty</code> attributes, optionally scoped by the <code>paramScope</code> attribute</p> <p>(RT EXPR)</p>   |
| paramName     | <p>The name of a JSP bean that is a String containing the value for the request parameter named by <code>paramId</code> (if <code>paramProperty</code> is not specified), or a JSP bean whose property getter is called to return a String (if <code>paramProperty</code> is specified). The JSP bean is constrained to the bean scope specified by the <code>paramScope</code> property, if it is specified.</p> <p>(RT EXPR)</p> |
| paramProperty | <p>The name of a property of the bean specified by the <code>paramName</code> attribute, whose return value must be a String containing the value of the request parameter (named by the <code>paramId</code> attribute) that will be dynamically added to this src URL.</p> <p>(RT EXPR)</p>  |
| paramScope    | <p>The scope within which to search for the bean specified by the <code>paramName</code> attribute. If not specified, all scopes are searched.</p> <p>(RT EXPR)</p>  |
| property      | <p>The name of a property of the bean specified by the <code>name</code> attribute, whose return value must be a <code>java.util.Map</code> containing the query parameters to be added to the src URL. You <b>must</b> specify the <code>name</code> attribute if you specify this attribute.</p> <p>(RT EXPR)</p>  |
| scope         | <p>The scope within which to search for the bean specified by the <code>name</code> attribute. If not specified, all scopes are searched.</p> <p>(RT EXPR)</p>   |
| src           | <p>The URL to which this image will be transferred from This image may be dynamically modified by the inclusion of query parameters, as described in the tag description. This value will be used unmodified (other than potential URL rewriting) as the value of the "src" attribute in the rendered tag. You <b>must</b> specify either the <code>page</code> attribute or the <code>src</code> attribute.</p> <p>(RT EXPR)</p>  |
| srcKey        | <p>The message key, in the message resources bundle named by the <code>bundle</code> attribute, of the String to be used as the URL of this image.</p> <p>(RT EXPR)</p>  |

|            |   |
|------------|---|
| style      | CSS styles to be applied to this element.<br>(RT EXPR)  |
| styleClass | CSS stylesheet class to be applied to this HTML element (renders a "class" attribute). (RT EXPR)  |
| styleId    | Identifier to be assigned to this HTML element (renders an "id" attribute). (RT EXPR)   |
| title      | The advisory title for this element.<br>(RT EXPR)   |
| titleKey   | The message resources key for the advisory title for this element.<br>(RT EXPR)   |
| usemap     | The name of the map as defined within this page for mapping hot-spot areas of this image.<br>(RT EXPR)  |
| vspace     | The amount of vertical spacing between the icon and the text, above and below.<br>(RT EXPR)   |
| width      | The width of the image being displayed. This parameter is very nice to specify (along with <code>height</code> ) to help the browser render the page faster.<br>(RT EXPR) |

[Back to top](#)

### javascript - Render JavaScript validation based on the validation rules loaded by the ValidatorPlugIn.

Render JavaScript validation based on the validation rules loaded by the `ValidatorPlugIn`. The set of validation rules that should be generated is based on the `formName` attribute passed in, which should match the name attribute of the form element in the xml file.

The `dynamicJavascript` and `staticJavascript` attributes default to `true`, but if `dynamicJavascript` is set to `true` and `staticJavascript` is set to `false` then only the dynamic JavaScript will be rendered. If `dynamicJavascript` is set to `false` and `staticJavascript` is set to `true` then only the static JavaScript will be rendered which can then be put in separate JSP page so the browser can cache the static JavaScript.

| Attribute Name    | Description  |
|-------------------|--|
| dynamicJavaScript | Whether or not to render the dynamic JavaScript. Defaults to <code>true</code> .   |
| formName          | The key (form name) to retrieve a specific set of validation rules.<br>(RT EXPR)   |
| htmlComment       | Whether or not to enclose the javascript with html comments. Defaults to <code>true</code> .<br>(RT EXPR)  |
| method            | The alternate JavaScript method name to be used instead of the of the default. The default is 'validate' concatenated in front of the key (form name) passed in (ex: validateRegistrationForm).<br>(RT EXPR) |
| page              | The current page of a set of validation rules if the page attribute for the field element in the xml file is in use.<br>(RT EXPR)  |
| src               | The src attribute's value when defining the html script element.<br>(RT EXPR)  |
| staticJavaScript  | Whether or not to render the static JavaScript. Defaults to <code>true</code> .  |

[Back to top](#)

### link - Render an HTML anchor or hyperlink

Renders an HTML `<a>` element as an anchor definition (if "linkName" is specified) or as a hyperlink to the specified URL. URL rewriting will be applied automatically, to maintain session state in the absence of cookies. The content displayed for this hyperlink will be taken from the body of this tag.

The base URL for this hyperlink is calculated based on which of the following attributes you specify (you must specify exactly one of them):

- *forward* - Use the value of this attribute as the name of a global `ActionForward` to be looked up, and use the application-relative or context-relative URI found there.
- *action* - Use the value of this attribute as the name of a `Action` to be looked up, and use the application-relative or context-relative URI found there.
- *href* - Use the value of this attribute unchanged.
- *page* - Use the value of this attribute as a application-relative URI, and generate a server-relative URI by including the context path and application prefix.

Normally, the hyperlink you specify with one of the attributes described in the previous paragraph will be left unchanged (other than URL rewriting if necessary). However, there are two ways you can append one or more dynamically defined query parameters to the hyperlink -- specify a single parameter with the `paramId` attribute (and its associated attributes to select the value), or specify the name (and optional `property`) attributes to select a `java.util.Map` bean that contains



one or more parameter ids and corresponding values.

To specify a single parameter, use the `paramId` attribute to define the name of the request parameter to be submitted. To specify the corresponding value, use one of the following approaches:

- *Specify only the `paramName` attribute* - The named JSP bean (optionally scoped by the value of the `paramScope` attribute) must identify a value that can be converted to a `String`.
- *Specify both the `paramName` and `paramProperty` attributes* - The specified property getter method will be called on the JSP bean identified by the `paramName` (and optional `paramScope`) attributes, in order to select a value that can be converted to a `String`.

If you prefer to specify a `java.util.Map` that contains all of the request parameters to be added to the hyperlink, use one of the following techniques:

- *Specify only the `name` attribute* - The named JSP bean (optionally scoped by the value of the `scope` attribute) must identify a `java.util.Map` containing the parameters.
- *Specify both `name` and `property` attributes* - The specified property getter method will be called on the bean identified by the `name` (and optional `scope`) attributes, in order to return the `java.util.Map` containing the parameters.

As the `Map` is processed, the keys are assumed to be the names of query parameters to be appended to the hyperlink. The value associated with each key must be either a `String` or a `String` array representing the parameter value(s), or an object whose `toString()` method will be called. If a `String` array is specified, more than one value for the same query parameter name will be created.

Additionally, you can request that the current transaction control token, if any, be included in the generated hyperlink by setting the `transaction` attribute to `true`. You can also request that an anchor ("`#xxx`") be added to the end of the URL that is created by any of the above mechanisms, by using the `anchor` attribute.

| Attribute Name         | Description   |
|------------------------|---|
| <code>accesskey</code> | The keyboard character used to move focus immediately to this element.<br>(RT EXPR)   |
| <code>action</code>    | Logical name of a <code>Action</code> that contains the actual content-relative URI of the destination of this transfer. This hyperlink may be dynamically modified by the inclusion of query parameters, as described in the tag description. You <b>must</b> specify exactly one of the <code>action</code> attribute, the <code>forward</code> attribute, the <code>href</code> attribute, the <code>linkName</code> attribute, or the <code>page</code> attribute.<br>(RT EXPR) |
| <code>anchor</code>    | Optional anchor tag (" <code>#xxx</code> ") to be added to the generated hyperlink. Specify this value <b>without</b> any <code>"#"</code> character.<br>(RT EXPR)  |

|            |  |
|------------|--|
| forward    | <p>Logical name of a global <code>ActionForward</code> that contains the actual content-relative URI of the destination of this transfer. This hyperlink may be dynamically modified by the inclusion of query parameters, as described in the tag description. You <b>must</b> specify exactly one of the <code>action</code> attribute, the <code>forward</code> attribute, the <code>href</code> attribute, the <code>linkName</code> attribute, or the <code>page</code> attribute.</p> <p>(RT EXPR)</p> |
| href       | <p>The URL to which this hyperlink will transfer control if activated. This hyperlink may be dynamically modified by the inclusion of query parameters, as described in the tag description. You <b>must</b> specify exactly one of the <code>action</code> attribute, the <code>forward</code> attribute, the <code>href</code> attribute, the <code>linkName</code> attribute, or the <code>page</code> attribute.</p> <p>(RT EXPR)</p>  |
| indexed    | <p>Valid only inside of <code>logic:iterate</code> tag. If <code>true</code> then indexed parameter with name from <code>indexId</code> attribute will be added to the query string. Indexed parameter looks like "index[32]". Number in brackets will be generated for every iteration and taken from ancestor <code>logic:iterate</code> tag. (RT EXPR)</p>  |
| indexId    | <p>By this attribute different name for the indexed parameter can be specified. Take a look to the "indexed" attribute for details. (RT EXPR)</p>  |
| linkName   | <p>The anchor name to be defined within this page, so that you can reference it with intra-page hyperlinks. In other words, the value specified here will render a "name" element in the generated anchor tag.</p> <p>(RT EXPR)</p>  |
| name       | <p>The name of a JSP bean that contains a <code>Map</code> representing the query parameters (if <code>property</code> is not specified), or a JSP bean whose property getter is called to return a <code>Map</code> (if <code>property</code> is specified).</p> <p>(RT EXPR)</p>   |
| onblur     | <p>JavaScript event handler that is executed when this element loses input focus.</p> <p>(RT EXPR)</p>   |
| onclick    | <p>JavaScript event handler that is executed when this element receives a mouse click.</p> <p>(RT EXPR)</p>  |
| ondblclick | <p>JavaScript event handler that is executed when this element receives a mouse double click.</p> <p>(RT EXPR)</p>   |
| onfocus    | <p>JavaScript event handler that is executed when this element receives input focus.</p> <p>(RT EXPR)</p>  |
| onkeydown  | <p>JavaScript event handler that is executed when this element receives a key down event.</p> <p>(RT EXPR)</p>   |

|             |   |
|-------------|---|
| onkeypress  | JavaScript event handler that is executed when this element receives a key press event.<br><br>(RT EXPR)  |
| onkeyup     | JavaScript event handler that is executed when this element receives a key up event.<br><br>(RT EXPR)   |
| onmousedown | JavaScript event handler that is executed when this element receives a mouse down event.<br><br>(RT EXPR)   |
| onmousemove | JavaScript event handler that is executed when this element receives a mouse move event.<br><br>(RT EXPR)   |
| onmouseout  | JavaScript event handler that is executed when this element receives a mouse out event.<br><br>(RT EXPR)  |
| onmouseover | JavaScript event handler that is executed when this element receives a mouse over event.<br><br>(RT EXPR)   |
| onmouseup   | JavaScript event handler that is executed when this element receives a mouse up event.<br><br>(RT EXPR)   |
| page        | The application-relative path (beginning with a "/" character) to which this hyperlink will transfer control if activated. This hyperlink may be dynamically modified by the inclusion of query parameters, as described in the tag description. You <b>must</b> specify exactly one of the <code>action</code> attribute, <code>forward</code> attribute, the <code>href</code> attribute, the <code>linkName</code> attribute, or the <code>page</code> attribute.<br><br>(RT EXPR) |
| paramId     | The name of the request parameter that will be dynamically added to the generated hyperlink. The corresponding value is defined by the <code>paramName</code> and (optional) <code>paramProperty</code> attributes, optionally scoped by the <code>paramScope</code> attribute<br><br>(RT EXPR)   |
| paramName   | The name of a JSP bean that is a String containing the value for the request parameter named by <code>paramId</code> (if <code>paramProperty</code> is not specified), or a JSP bean whose property getter is called to return a String (if <code>paramProperty</code> is specified). The JSP bean is constrained to the bean scope specified by the <code>paramScope</code> property, if it is specified.<br><br>(RT EXPR)   |

|               |   |
|---------------|---|
| paramProperty | <p>The name of a property of the bean specified by the <code>paramName</code> attribute, whose return value must be a <code>String</code> containing the value of the request parameter (named by the <code>paramId</code> attribute) that will be dynamically added to this hyperlink.</p> <p>(RT EXPR)</p>          |
| paramScope    | <p>The scope within which to search for the bean specified by the <code>paramName</code> attribute. If not specified, all scopes are searched.</p> <p>(RT EXPR)</p>   |
| property      | <p>The name of a property of the bean specified by the <code>name</code> attribute, whose return value must be a <code>java.util.Map</code> containing the query parameters to be added to the hyperlink. You <b>must</b> specify the <code>name</code> attribute if you specify this attribute.</p> <p>(RT EXPR)</p> |
| scope         | <p>The scope within which to search for the bean specified by the <code>name</code> attribute. If not specified, all scopes are searched.</p> <p>(RT EXPR)</p>  |
| style         | <p>CSS styles to be applied to this element.</p> <p>(RT EXPR)</p>   |
| styleClass    | <p>CSS stylesheet class to be applied to this HTML element (renders a "class" attribute). (RT EXPR)</p>   |
| styleId       | <p>Identifier to be assigned to this HTML element (renders an "id" attribute). (RT EXPR)</p>  |
| tabindex      | <p>The tab order (ascending positive integers) for this element.</p> <p>(RT EXPR)</p>   |
| target        | <p>The window target in which the resource requested by this hyperlink will be displayed, for example in a framed presentation.</p> <p>(RT EXPR)</p>  |
| title         | <p>The advisory title for this hyperlink.</p> <p>(RT EXPR)</p>  |
| titleKey      | <p>The message resources key for the advisory title for this element.</p> <p>(RT EXPR)</p>  |
| transaction   | <p>If set to <code>true</code>, any current transaction control token will be included in the generated hyperlink, so that it will pass an <code>isTokenValid()</code> test in the receiving Action.</p> <p>(RT EXPR)</p>   |

[Back to top](#)

**messages** - Conditionally display a set of accumulated messages.

Displays a set of messages prepared by a business logic component and stored as an `ActionMessages` object, `ActionErrors` object, a `String`, or a `String` array in request scope. If such a bean is not found, nothing will be rendered.

In order to use this tag successfully, you must have defined an application scope `MessageResources` bean under the default attribute name.

| Attribute Name | Description   |
|----------------|---|
| bundle         | The servlet context attribute key for the <code>MessageResources</code> instance to use. If not specified, defaults to the application resources configured for our action servlet. (RT EXPR)   |
| footer         | This value is an optional message resource key that will be printed after the iteration of messages has finished. (RT EXPR)   |
| header         | This value is an optional message resource key that will be printed before the iteration of messages begins. (RT EXPR)  |
| id             | The name of a page scope JSP bean that will contain the current element of the collection of messages on each iteration, if it is not <code>null</code> . (REQUIRED) (RT EXPR)  |
| locale         | The session attribute key for the <code>Locale</code> used to select messages to be displayed. If not specified, defaults to the Struts standard value. (RT EXPR)   |
| message        | By default the tag will retrieve the request scope bean it will iterate over from the <code>Action.ERROR_KEY</code> constant string, but if this attribute is set to 'true' the request scope bean will be retrieved from the <code>Action.MESSAGE_KEY</code> constant string. Also if this is set to 'true', any value assigned to the name attribute will be ignored. (RT EXPR) |
| name           | Name of the request scope bean under which our messages have been stored. If not present, the name specified by the <code>Action.ERROR_KEY</code> constant string will be used. (RT EXPR)   |
| property       | Name of the property for which messages should be displayed. If not specified, all messages (regardless of property) are displayed. (RT EXPR)   |

[Back to top](#)

**multibox** - Render A Checkbox Input Field

Renders an HTML `<input>` element of type `checkbox`, whose "checked" status is initialized based on whether the specified value matches one of the elements of the underlying property's array of current values. This element is useful when you have large numbers of checkboxes, and prefer to combine the values into a single array-valued property instead of multiple boolean properties. This tag is only valid when nested inside a form tag body.

**WARNING:** In order to correctly recognize cases where none of the associated checkboxes are selected, the `ActionForm` bean associated with this form must include a statement setting the corresponding array to zero length in the `reset()` method.

The value to be returned to the server, if this checkbox is selected, must be defined by one of the following methods:

- Specify a `value` attribute, whose contents will be used literally as the value to be returned.
- Specify no `value` attribute, and the nested body content of this tag will be used as the value to be returned.

| Attribute Name           | Description   |
|--------------------------|---|
| <code>accesskey</code>   | The keyboard character used to move focus immediately to this element. (RT EXPR)  |
| <code>alt</code>         | The alternate text for this element.<br>(RT EXPR)   |
| <code>altKey</code>      | The message resources key of the alternate text for this element.<br>(RT EXPR)  |
| <code>disabled</code>    | Set to <code>true</code> if this input field should be disabled. (RT EXPR)  |
| <code>name</code>        | The attribute name of the bean whose properties are consulted when rendering the current value of this input field. If not specified, the bean associated with the form tag we are nested within is utilized. (RT EXPR) |
| <code>onblur</code>      | JavaScript event handler executed when this element loses input focus. (RT EXPR)  |
| <code>onchange</code>    | JavaScript event handler executed when this element loses input focus and its value has changed. (RT EXPR)  |
| <code>onclick</code>     | JavaScript event handler executed when this element receives a mouse click. (RT EXPR)   |
| <code>ondblclick</code>  | JavaScript event handler executed when this element receives a mouse double click. (RT EXPR)  |
| <code>onfocus</code>     | JavaScript event handler executed when this element receives input focus. (RT EXPR)   |
| <code>onkeydown</code>   | JavaScript event handler executed when this element has focus and a key is depressed. (RT EXPR)   |
| <code>onkeypress</code>  | JavaScript event handler executed when this element has focus and a key is depressed and released. (RT EXPR)  |
| <code>onkeyup</code>     | JavaScript event handler executed when this element has focus and a key is released. (RT EXPR)  |
| <code>onmousedown</code> | JavaScript event handler executed when this element is under the mouse pointer and a mouse button is depressed. (RT EXPR)   |
| <code>onmousemove</code> | JavaScript event handler executed when this element is under the mouse pointer and the pointer is moved. (RT EXPR)  |
| <code>onmouseout</code>  | JavaScript event handler executed when this element was under the mouse pointer but the pointer was moved outside the element. (RT EXPR)  |
| <code>onmouseover</code> | JavaScript event handler executed when this element was not under the mouse pointer but the pointer is moved inside the element. (RT EXPR)  |
| <code>onmouseup</code>   | JavaScript event handler executed when this element is under the mouse pointer and a mouse button is released. (RT EXPR)  |

|            |  |
|------------|--|
| property   | Name of the request parameter that will be included with this submission, set to the specified value. (REQUIRED) (RT EXPR) |
| style      | CSS styles to be applied to this HTML element. (RT EXPR)   |
| styleClass | CSS stylesheet class to be applied to this HTML element (renders a "class" attribute). (RT EXPR)                           |
| styleId    | Identifier to be assigned to this HTML element (renders an "id" attribute). (RT EXPR)                                      |
| tabindex   | The tab order (ascending positive integers) for this element. (RT EXPR)  |
| title      | The advisory title for this element.<br>(RT EXPR)  |
| titleKey   | The message resources key for the advisory title for this element.<br>(RT EXPR)  |
| value      | The value to be transmitted if this checkbox is checked when the form is submitted. (RT EXPR)                              |

[Back to top](#)

#### option - Render A Select Option

Render an HTML `<option>` element, representing one of the choices for an enclosing `<select>` element. The text displayed to the user comes from either the body of this tag, or from a message string looked up based on the `bundle`, `locale`, and `key` attributes.

If the value of the corresponding bean property matches the specified value, this option will be marked selected. This tag is only valid when nested inside a `<html:select>` tag body.

| Attribute Name | Description  |
|----------------|--|
| bundle         | The servlet context attributes key for the MessageResources instance to use. If not specified, defaults to the application resources configured for our action servlet. (RT EXPR)  |
| disabled       | Set to <code>true</code> if this option should be disabled. (RT EXPR)  |
| key            | If specified, defines the message key to be looked up in the resource bundle specified by <code>bundle</code> for the text displayed to the user for this option. If not specified, the text to be displayed is taken from the body content of this tag. (RT EXPR) |
| locale         | The session attributes key for the Locale instance to use for looking up the message specified by the <code>key</code> attribute. If not specified, uses the standard Struts session attribute name. (RT EXPR)   |
| style          | CSS styles to be applied to this HTML element. (RT EXPR)   |
| styleClass     | CSS stylesheet class to be applied to this HTML element (renders a "class" attribute). (RT EXPR)   |
| styleId        | Identifier to be assigned to this HTML element (renders an "id" attribute). (RT EXPR)  |



|       |  |
|-------|--|
| value | Value to be submitted for this field if this option is selected by the user.<br>(REQUIRED) (RT EXPR) |
|-------|--|

[Back to top](#)

## options - Render a Collection of Select Options

Renders a set of HTML `<option>` elements, representing possible choices for a `<select>` element. This tag can be used multiple times within a single `<html:select>` element, either in conjunction with or instead of one or more `<html:option>` or `<html:optionsCollection>` elements.

This tag operates in one of two major modes, depending on whether or not the `collection` attribute is specified. If the `collection` attribute is included, the following rules apply:

- The **collection** attribute is interpreted as the name of a JSP bean, in some scope, that itself represents a collection of individual beans, one per option value to be rendered.
- The **property** attribute is interpreted as the name of a property of the individual beans included in the collection, and is used to retrieve the value that will be returned to the server if this option is selected.
- The **labelProperty** attribute is interpreted as the name of a property of the individual beans included in the collection, and is used to retrieve the label that will be displayed to the user for this option. If the `labelProperty` attribute is not specified, the property named by the `property` attribute will be used to select both the value returned to the server and the label displayed to the user for this option.

If the `collection` attribute is not specified, the rules described in the remainder of this section apply.

The collection of values actually selected depends on the presence or absence of the `name` and `property` attributes. The following combinations are allowed:

- *Only name is specified* - The value of this attribute is the name of a JSP bean in some scope that is the collection.
- *Only property is specified* - The value of this attribute is the name of a property of the ActionForm bean associated with our form, which will return the collection.
- *Both name and property are specified* - The value of the `name` attribute identifies a JSP bean in some scope. The value of the `property` attribute is the name of some property of that bean which will return the collection.

The collection of labels displayed to the user can be the same as the option values themselves, or can be different, depending on the presence or absence of the `labelName` and `labelProperty` attributes. If this feature is used, the collection of labels must contain the same number of elements as the corresponding collection of values. The following combinations are allowed:

- *Neither labelName nor labelProperty is specified* - The labels will be the same as the option values themselves.
- *Only labelName is specified* - The value of this attribute is the name of a JSP bean in some scope that is the collection.
- *Only labelProperty is specified* - The value of this attribute is the name of a property of the ActionForm bean associated with our form, which will return the collection.
- *Both labelName and labelProperty are specified* - The value of the `labelName` attribute identifies a JSP bean in some scope. The value of the `labelProperty` attribute



is the name of some property of that bean which will return the collection.

Note that this tag does not support a `styleId` attribute, as it would have to apply the value to all the `option` elements created by this element, which would mean that more than one `id` element might have the same value, which the HTML specification says is illegal.

| Attribute Name | Description  |
|----------------|--|
| collection     | Name of the JSP bean (in some scope) which is itself a Collection of other beans, each of which has properties named by the "property" and "labelProperty" attributes that are used to retrieve the value and label for each option, respectively. (RT EXPR) |
| filter         | Set to <code>false</code> if you do NOT want the option labels filtered for sensitive characters in HTML. By default, such values are filtered. (RT EXPR)  |
| labelName      | Name of the JSP bean (in some scope) containing the collection of labels to be displayed to the user for these options. (RT EXPR)  |
| labelProperty  | Property of the form bean, or the bean specified by the <code>labelName</code> attribute, that will return the collection of labels to be displayed to the user for these options. (RT EXPR)   |
| name           | Name of the JSP bean (in some scope) containing the collection of values to be returned to the server for these options. If not specified, the form bean associated with our form is assumed. (RT EXPR)  |
| property       | Property of the form bean, or the bean specified by the <code>name</code> attribute, that will return the collection of values to returned to the server for these options. (RT EXPR)  |
| style          | CSS styles to be applied to this HTML element. (RT EXPR)   |
| styleClass     | CSS stylesheet class to be applied to this HTML element (renders a "class" attribute). (RT EXPR)   |

[Back to top](#)

### optionsCollection - Render a Collection of Select Options

Renders a set of HTML `<option>` elements, representing possible choices for a `<select>` element. This tag can be used multiple times within a single `<html:select>` element, either in conjunction with or instead of one or more `<html:option>` or `<html:options>` elements.

This tag operates on a collection of beans, where each bean has a **label** property and a **value** property. The actual names of these properties can be configured using the `label` and `value` attributes of this tag.

This tag differs from the `<html:options>` tag in that it makes more consistent use of the `name` and `property` attributes, and allows the collection to be more easily obtained from the enclosing form bean.

Note that this tag does not support a `styleId` attribute, as it would have to apply the value to all the `option` elements created by this element, which would mean that more than one `id` element might have the same value, which the HTML specification says is illegal.

| Attribute Name | Description   |
|----------------|---|
| filter         | Set to <code>false</code> if you do NOT want the option labels filtered for sensitive characters in HTML. By default, such values are filtered. (RT EXPR)   |
| label          | The property of the bean within the collection which represents the label to be rendered for each option. Defaults to "label". (RT EXPR)  |
| name           | The attribute name of the bean whose properties are consulted when rendering the current value of this input field. If not specified, the bean associated with the form tag we are nested within is utilized. (RT EXPR) |
| property       | The property of the form bean, or the bean specified by the name attribute, that will return the collection of objects to be rendered for these options. (REQUIRED) (RT EXPR)   |
| style          | CSS styles to be applied to this HTML element. (RT EXPR)  |
| styleClass     | CSS stylesheet class to be applied to this HTML element (renders a "class" attribute). (RT EXPR)  |
| value          | The property of the bean within the collection which represents the value to be rendered for each option. Defaults to "value". (RT EXPR)  |

[Back to top](#)

### password - Render A Password Input Field

Renders an HTML `<input>` element of type `password`, populated from the specified value or the specified property of the bean associated with our current form. This tag is only valid when nested inside a form tag body.

| Attribute Name | Description   |
|----------------|---|
| accesskey      | The keyboard character used to move focus immediately to this element. (RT EXPR)  |
| alt            | The alternate text for this element.<br>(RT EXPR)   |
| altKey         | The message resources key of the alternate text for this element.<br>(RT EXPR)  |
| disabled       | Set to <code>true</code> if this input field should be disabled. (RT EXPR)  |
| indexed        | Valid only inside of <code>logic:iterate</code> tag. If <code>true</code> then name of the html tag will be rendered as "id[34].propertyName". Number in brackets will be generated for every iteration and taken from ancestor <code>logic:iterate</code> tag. (RT EXPR) |
| maxlength      | Maximum number of input characters to accept. [No limit] (RT EXPR)  |
| name           | The attribute name of the bean whose properties are consulted when rendering the current value of this input field. If not specified, the bean associated with the form tag we are nested within is utilized. (RT EXPR)   |
| onblur         | JavaScript event handler executed when this element loses input focus. (RT EXPR)  |

|             |   |
|-------------|---|
| onchange    | JavaScript event handler executed when this element loses input focus and its value has changed. (RT EXPR)  |
| onclick     | JavaScript event handler executed when this element receives a mouse click. (RT EXPR)   |
| ondblclick  | JavaScript event handler executed when this element receives a mouse double click. (RT EXPR)  |
| onfocus     | JavaScript event handler executed when this element receives input focus. (RT EXPR)   |
| onkeydown   | JavaScript event handler executed when this element has focus and a key is depressed. (RT EXPR)   |
| onkeypress  | JavaScript event handler executed when this element has focus and a key is depressed and released. (RT EXPR)  |
| onkeyup     | JavaScript event handler executed when this element has focus and a key is released. (RT EXPR)  |
| onmousedown | JavaScript event handler executed when this element is under the mouse pointer and a mouse button is depressed. (RT EXPR)   |
| onmousemove | JavaScript event handler executed when this element is under the mouse pointer and the pointer is moved. (RT EXPR)  |
| onmouseout  | JavaScript event handler executed when this element was under the mouse pointer but the pointer was moved outside the element. (RT EXPR)  |
| onmouseover | JavaScript event handler executed when this element was not under the mouse pointer but the pointer is moved inside the element. (RT EXPR)  |
| onmouseup   | JavaScript event handler executed when this element is under the mouse pointer and a mouse button is released. (RT EXPR)  |
| property    | Name of the request parameter that will be included with this submission, set to the specified value. (REQUIRED) (RT EXPR)  |
| readonly    | Set to <code>true</code> if this input field should be read only. (RT EXPR)   |
| redisplay   | Boolean flag indicating whether or not existing values will be redisplayed if they exist. Even though the redisplayed value will be shown as asterisks on the visible HTML page, the cleartext of the actual password value will be visible though the "Show Page Source" menu option of the client browser. You may wish to set this value to <code>false</code> on login pages. Defaults to <code>true</code> for consistency with all other form tags that redisplay their contents. (RT EXPR) |
| size        | Number of character positions to allocate. [Browser default] (RT EXPR)  |
| style       | CSS styles to be applied to this HTML element. (RT EXPR)  |
| styleClass  | CSS stylesheet class to be applied to this HTML element (renders a "class" attribute). (RT EXPR)  |
| styleId     | Identifier to be assigned to this HTML element (renders an "id" attribute). (RT EXPR)   |
| tabindex    | The tab order (ascending positive integers) for this element. (RT EXPR)   |
| title       | The advisory title for this element.<br>(RT EXPR)   |
| titleKey    | The message resources key for the advisory title for this element.<br>(RT EXPR)   |

|       |   |
|-------|---|
| value | Value of the label to be placed on this button. This value will also be submitted as the value of the specified request parameter. [Body of this tag (if any), or "Cancel"] (RT EXPR) |
|-------|---|

[Back to top](#)

### radio - Render A Radio Button Input Field

Renders an HTML `<input>` element of type `radio`, populated from the specified property of the bean associated with our current form. This tag is only valid when nested inside a form tag body.

If an iterator is used to render a series of radio tags, the `idName` attribute may be used to specify the name of the bean exposed by the iterator. In this case, the `value` attribute is used as the name of a property on the `idName` bean that returns the value of the radio tag in this iteration.

| Attribute Name | Description   |
|----------------|---|
| accesskey      | The keyboard character used to move focus immediately to this element. (RT EXPR)  |
| alt            | The alternate text for this element.<br>(RT EXPR)   |
| altKey         | The message resources key of the alternate text for this element.<br>(RT EXPR)  |
| disabled       | Set to <code>true</code> if this input field should be disabled. (RT EXPR)  |
| idName         | Since: Struts 1.1<br><br>Name of the bean (in some scope) that will return the value of the radio tag. Usually exposed by an iterator. When the <code>idName</code> attribute is present, the <code>value</code> attribute is used as the name of the property on the <code>idName</code> bean that will return the value of the radio tag for this iteration.<br><br>(RT EXPR) |
| indexed        | Valid only inside of <code>logic:iterate</code> tag. If <code>true</code> then name of the html tag will be rendered as <code>"id[34].propertyName"</code> . Number in brackets will be generated for every iteration and taken from ancestor <code>logic:iterate</code> tag. (RT EXPR)   |
| name           | The attribute name of the bean whose properties are consulted when rendering the current value of this input field. If not specified, the bean associated with the form tag we are nested within is utilized. (RT EXPR)   |
| onblur         | JavaScript event handler executed when this element loses input focus. (RT EXPR)  |
| onchange       | JavaScript event handler executed when this element loses input focus and its value has changed. (RT EXPR)  |
| onclick        | JavaScript event handler executed when this element receives a mouse click. (RT EXPR)   |
| ondblclick     | JavaScript event handler executed when this element receives a mouse double click. (RT EXPR)  |

|             |  |
|-------------|--|
| onfocus     | JavaScript event handler executed when this element receives input focus. (RT EXPR)  |
| onkeydown   | JavaScript event handler executed when this element has focus and a key is depressed. (RT EXPR)  |
| onkeypress  | JavaScript event handler executed when this element has focus and a key is depressed and released. (RT EXPR)                               |
| onkeyup     | JavaScript event handler executed when this element has focus and a key is released. (RT EXPR)   |
| onmousedown | JavaScript event handler executed when this element is under the mouse pointer and a mouse button is depressed. (RT EXPR)                  |
| onmousemove | JavaScript event handler executed when this element is under the mouse pointer and the pointer is moved. (RT EXPR)                         |
| onmouseout  | JavaScript event handler executed when this element was under the mouse pointer but the pointer was moved outside the element. (RT EXPR)   |
| onmouseover | JavaScript event handler executed when this element was not under the mouse pointer but the pointer is moved inside the element. (RT EXPR) |
| onmouseup   | JavaScript event handler executed when this element is under the mouse pointer and a mouse button is released. (RT EXPR)                   |
| property    | The corresponding bean property for this radio tag. (REQUIRED) (RT EXPR)   |
| style       | CSS styles to be applied to this HTML element. (RT EXPR)   |
| styleClass  | CSS stylesheet class to be applied to this HTML element (renders a "class" attribute). (RT EXPR)   |
| styleId     | Identifier to be assigned to this HTML element (renders an "id" attribute). (RT EXPR)  |
| tabindex    | The tab order (ascending positive integers) for this element. (RT EXPR)  |
| title       | The advisory title for this element.<br>(RT EXPR)  |
| titleKey    | The message resources key for the advisory title for this element.<br>(RT EXPR)  |
| value       | The value of the radio tag. (REQUIRED) (RT EXPR)   |

[Back to top](#)

## reset - Render A Reset Button Input Field

Renders an HTML <input> element of type reset.

| Attribute Name | Description  |
|----------------|--|
| accesskey      | The keyboard character used to move focus immediately to this element. (RT EXPR)   |
| alt            | The alternate text for this element.<br>(RT EXPR)  |
| altKey         | The message resources key of the alternate text for this element.<br>(RT EXPR)   |
| disabled       | Set to <code>true</code> if this input field should be disabled. (RT EXPR)   |
| onblur         | JavaScript event handler executed when this element loses input focus. (RT EXPR)   |
| onchange       | JavaScript event handler executed when this element loses input focus and its value has changed. (RT EXPR)                                 |
| onclick        | JavaScript event handler executed when this element receives a mouse click. (RT EXPR)  |
| ondblclick     | JavaScript event handler executed when this element receives a mouse double click. (RT EXPR)   |
| onfocus        | JavaScript event handler executed when this element receives input focus. (RT EXPR)  |
| onkeydown      | JavaScript event handler executed when this element has focus and a key is depressed. (RT EXPR)  |
| onkeypress     | JavaScript event handler executed when this element has focus and a key is depressed and released. (RT EXPR)                               |
| onkeyup        | JavaScript event handler executed when this element has focus and a key is released. (RT EXPR)   |
| onmousedown    | JavaScript event handler executed when this element is under the mouse pointer and a mouse button is depressed. (RT EXPR)                  |
| onmousemove    | JavaScript event handler executed when this element is under the mouse pointer and the pointer is moved. (RT EXPR)                         |
| onmouseout     | JavaScript event handler executed when this element was under the mouse pointer but the pointer was moved outside the element. (RT EXPR)   |
| onmouseover    | JavaScript event handler executed when this element was not under the mouse pointer but the pointer is moved inside the element. (RT EXPR) |
| onmouseup      | JavaScript event handler executed when this element is under the mouse pointer and a mouse button is released. (RT EXPR)                   |
| property       | Name of the input field that will be generated. (RT EXPR)  |
| style          | CSS styles to be applied to this HTML element. (RT EXPR)   |
| styleClass     | CSS stylesheet class to be applied to this HTML element (renders a "class" attribute). (RT EXPR)   |
| styleId        | Identifier to be assigned to this HTML element (renders an "id" attribute). (RT EXPR)  |
| tabindex       | The tab order (ascending positive integers) for this element. (RT EXPR)  |

|          |   |
|----------|---|
| title    | The advisory title for this element.<br>(RT EXPR)   |
| titleKey | The message resources key for the advisory title for this element.<br>(RT EXPR)                   |
| value    | Value of the label to be placed on this button. [Body of this tag (if any), or "Reset"] (RT EXPR) |

[Back to top](#)

## rewrite - Render an URI

Renders a request URI based on exactly the same rules as the [link](#) tag does, but without creating the <a> hyperlink. This value is useful when you want to generate a string constant for use by a JavaScript procedure.

| Attribute Name | Description  |
|----------------|--|
| anchor         | Optional anchor tag ("#xxx") to be added to the generated hyperlink. Specify this value <b>without</b> any "#" character.<br>(RT EXPR)   |
| forward        | Logical name of a global <code>ActionForward</code> that contains the actual content-relative URI of the destination of this transfer. This hyperlink may be dynamically modified by the inclusion of query parameters, as described in the tag description. You <b>must</b> specify exactly one of the <code>action</code> attribute, the <code>forward</code> attribute, the <code>href</code> attribute, or the <code>page</code> attribute.<br>(RT EXPR) |
| href           | The URL to which this hyperlink will transfer control if activated. This hyperlink may be dynamically modified by the inclusion of query parameters, as described in the tag description. You <b>must</b> specify exactly one of the <code>action</code> attribute, the <code>forward</code> attribute, the <code>href</code> attribute, or the <code>page</code> attribute.<br>(RT EXPR)  |
| name           | The name of a JSP bean that contains a <code>Map</code> representing the query parameters (if <code>property</code> is not specified), or a JSP bean whose property getter is called to return a <code>Map</code> (if <code>property</code> is specified).<br>(RT EXPR)  |
| page           | The application-relative path (beginning with a "/" character) to which this hyperlink will transfer control if activated. This hyperlink may be dynamically modified by the inclusion of query parameters, as described in the tag description. You <b>must</b> specify exactly one of the <code>action</code> attribute, the <code>forward</code> attribute, the <code>href</code> attribute, or the <code>page</code> attribute.<br>(RT EXPR)             |

|               |  |
|---------------|--|
| paramId       | <p>The name of the request parameter that will be dynamically added to the generated hyperlink. The corresponding value is defined by the <code>paramName</code> and (optional) <code>paramProperty</code> attributes, optionally scoped by the <code>paramScope</code> attribute</p> <p>(RT EXPR)</p>   |
| paramName     | <p>The name of a JSP bean that is a String containing the value for the request parameter named by <code>paramId</code> (if <code>paramProperty</code> is not specified), or a JSP bean whose property getter is called to return a String (if <code>paramProperty</code> is specified). The JSP bean is constrained to the bean scope specified by the <code>paramScope</code> property, if it is specified.</p> <p>(RT EXPR)</p> |
| paramProperty | <p>The name of a property of the bean specified by the <code>paramName</code> attribute, whose return value must be a String containing the value of the request parameter (named by the <code>paramId</code> attribute) that will be dynamically added to this hyperlink.</p> <p>(RT EXPR)</p>  |
| paramScope    | <p>The scope within which to search for the bean specified by the <code>paramName</code> attribute. If not specified, all scopes are searched.</p> <p>(RT EXPR)</p>  |
| property      | <p>The name of a property of the bean specified by the <code>name</code> attribute, whose return value must be a <code>java.util.Map</code> containing the query parameters to be added to the hyperlink. You <b>must</b> specify the <code>name</code> attribute if you specify this attribute.</p> <p>(RT EXPR)</p>  |
| scope         | <p>The scope within which to search for the bean specified by the <code>name</code> attribute. If not specified, all scopes are searched.</p> <p>(RT EXPR)</p>   |
| transaction   | <p>If set to <code>true</code>, any current transaction control token will be included in the generated hyperlink, so that it will pass an <code>isTokenValid()</code> test in the receiving Action.</p> <p>(RT EXPR)</p>  |

[Back to top](#)

## select - Render A Select Element



Renders an HTML `<select>` element, associated with a bean property specified by our attributes. This tag is only valid when nested inside a form tag body.

This tag operates in two modes, depending upon the state of the `multiple` attribute, which affects the data type of the associated property you should use:

- *multiple="true" IS NOT selected* - The corresponding property should be a scalar value of any supported data type.
- *multiple="true" IS selected* - The corresponding property should be an array of any supported data type.

**WARNING:** In order to correctly recognize cases where no selection at all is made, the `ActionForm` bean associated with this form must include a statement resetting the scalar property to a default value (if `multiple` is not set), or the array property to zero length (if `multiple` is set) in the `reset()` method.

| Attribute Name | Description   |
|----------------|---|
| alt            | The alternate text for this element.<br>(RT EXPR)   |
| altKey         | The message resources key of the alternate text for this element.<br>(RT EXPR)  |
| disabled       | Set to <code>true</code> if this input field should be disabled. (RT EXPR)  |
| indexed        | Valid only inside of <code>logic:iterate</code> tag. If <code>true</code> then name of the html tag will be rendered as <code>"id[34].propertyName"</code> . Number in brackets will be generated for every iteration and taken from ancestor <code>logic:iterate</code> tag. (RT EXPR) |
| multiple       | If set to any arbitrary value, the rendered select element will support multiple selections. (RT EXPR)  |
| name           | The attribute name of the bean whose properties are consulted to determine which option should be pre-selected when rendering this input field. If not specified, the bean associated with the enclosing <code>&lt;html:form&gt;</code> tag is utilized. (RT EXPR)                      |
| onblur         | JavaScript event handler executed when this element loses input focus. (RT EXPR)  |
| onchange       | JavaScript event handler executed when this element loses input focus and its value has changed. (RT EXPR)  |
| onclick        | JavaScript event handler executed when this element receives a mouse click. (RT EXPR)   |
| ondblclick     | JavaScript event handler executed when this element receives a mouse double click. (RT EXPR)  |
| onfocus        | JavaScript event handler executed when this element receives input focus. (RT EXPR)   |
| onkeydown      | JavaScript event handler executed when this element has focus and a key is depressed. (RT EXPR)   |
| onkeypress     | JavaScript event handler executed when this element has focus and a key is depressed and released. (RT EXPR)  |

|             |  |
|-------------|--|
| onkeyup     | JavaScript event handler executed when this element has focus and a key is released. (RT EXPR)   |
| onmousedown | JavaScript event handler executed when this element is under the mouse pointer and a mouse button is depressed. (RT EXPR)                  |
| onmousemove | JavaScript event handler executed when this element is under the mouse pointer and the pointer is moved. (RT EXPR)                         |
| onmouseout  | JavaScript event handler executed when this element was under the mouse pointer but the pointer was moved outside the element. (RT EXPR)   |
| onmouseover | JavaScript event handler executed when this element was not under the mouse pointer but the pointer is moved inside the element. (RT EXPR) |
| onmouseup   | JavaScript event handler executed when this element is under the mouse pointer and a mouse button is released. (RT EXPR)                   |
| property    | Name of the request parameter that will be included with this submission, set to the specified value. (REQUIRED) (RT EXPR)                 |
| size        | The number of available options displayed at one time. (RT EXPR)   |
| style       | CSS styles to be applied to this HTML element. (RT EXPR)   |
| styleClass  | CSS stylesheet class to be applied to this HTML element (renders a "class" attribute). (RT EXPR)   |
| styleId     | Identifier to be assigned to this HTML element (renders an "id" attribute). (RT EXPR)  |
| tabindex    | The tab order (ascending positive integers) for this element. (RT EXPR)  |
| title       | The advisory title for this element.<br>(RT EXPR)  |
| titleKey    | The message resources key for the advisory title for this element.<br>(RT EXPR)  |
| value       | The value to compare with for marking an option selected. (RT EXPR)  |

[Back to top](#)

### submit - Render A Submit Button

Renders an HTML `<input>` element of type `submit`.

If a graphical button is needed (a button with an image), then the [image](#) tag is more appropriate.

| Attribute Name | Description  |
|----------------|--|
| accesskey      | The keyboard character used to move focus immediately to this element. (RT EXPR)   |
| alt            | The alternate text for this element.<br>(RT EXPR)  |
| altKey         | The message resources key of the alternate text for this element.<br>(RT EXPR)   |
| disabled       | Set to <code>true</code> if this input field should be disabled. (RT EXPR)   |
| indexed        | Valid only inside of <code>logic:iterate</code> tag. If <code>true</code> then name of the html tag will be rendered as "propertyName[34]". Number in brackets will be generated for every iteration and taken from ancestor <code>logic:iterate</code> tag. (RT EXPR) |
| onblur         | JavaScript event handler executed when this element loses input focus. (RT EXPR)   |
| onchange       | JavaScript event handler executed when this element loses input focus and its value has changed. (RT EXPR)   |
| onclick        | JavaScript event handler executed when this element receives a mouse click. (RT EXPR)  |
| ondblclick     | JavaScript event handler executed when this element receives a mouse double click. (RT EXPR)   |
| onfocus        | JavaScript event handler executed when this element receives input focus. (RT EXPR)  |
| onkeydown      | JavaScript event handler executed when this element has focus and a key is depressed. (RT EXPR)  |
| onkeypress     | JavaScript event handler executed when this element has focus and a key is depressed and released. (RT EXPR)   |
| onkeyup        | JavaScript event handler executed when this element has focus and a key is released. (RT EXPR)   |
| onmousedown    | JavaScript event handler executed when this element is under the mouse pointer and a mouse button is depressed. (RT EXPR)  |
| onmousemove    | JavaScript event handler executed when this element is under the mouse pointer and the pointer is moved. (RT EXPR)   |
| onmouseout     | JavaScript event handler executed when this element was under the mouse pointer but the pointer was moved outside the element. (RT EXPR)   |
| onmouseover    | JavaScript event handler executed when this element was not under the mouse pointer but the pointer is moved inside the element. (RT EXPR)   |
| onmouseup      | JavaScript event handler executed when this element is under the mouse pointer and a mouse button is released. (RT EXPR)   |
| property       | Name of the request parameter that will be included with this submission, set to the specified value. (RT EXPR)  |
| style          | CSS styles to be applied to this HTML element. (RT EXPR)   |
| styleClass     | CSS stylesheet class to be applied to this HTML element (renders a "class" attribute). (RT EXPR)   |

|          |   |
|----------|---|
| styleId  | Identifier to be assigned to this HTML element (renders an "id" attribute). (RT EXPR) |
| tabindex | The tab order (ascending positive integers) for this element. (RT EXPR)               |
| title    | The advisory title for this element.<br>(RT EXPR)                                     |
| titleKey | The message resources key for the advisory title for this element.<br>(RT EXPR)       |
| value    | The value of the button label. (RT EXPR)  |

[Back to top](#)

### text - Render An Input Field of Type text

Render an input button of type text. This tag is only valid when nested inside a form tag body.

| Attribute Name | Description   |
|----------------|---|
| accesskey      | The keyboard character used to move focus immediately to this element. (RT EXPR)  |
| alt            | The alternate text for this element.<br>(RT EXPR)   |
| altKey         | The message resources key of the alternate text for this element.<br>(RT EXPR)  |
| disabled       | Set to <code>true</code> if this input field should be disabled. (RT EXPR)  |
| indexed        | Valid only inside of <code>logic:iterate</code> tag. If <code>true</code> then name of the html tag will be rendered as "id[34].propertyName". Number in brackets will be generated for every iteration and taken from ancestor <code>logic:iterate</code> tag. (RT EXPR) |
| maxlength      | Maximum number of input characters to accept. [No limit] (RT EXPR)  |
| name           | The attribute name of the bean whose properties are consulted when rendering the current value of this input field. If not specified, the bean associated with the form tag we are nested within is utilized. (RT EXPR)   |
| onblur         | JavaScript event handler executed when this element loses input focus. (RT EXPR)  |
| onchange       | JavaScript event handler executed when this element loses input focus and its value has changed. (RT EXPR)  |
| onclick        | JavaScript event handler executed when this element receives a mouse click. (RT EXPR)   |
| ondblclick     | JavaScript event handler executed when this element receives a mouse double click. (RT EXPR)  |
| onfocus        | JavaScript event handler executed when this element receives input focus. (RT EXPR)   |

|             |  |
|-------------|--|
| onkeydown   | JavaScript event handler executed when this element has focus and a key is depressed. (RT EXPR)  |
| onkeypress  | JavaScript event handler executed when this element has focus and a key is depressed and released. (RT EXPR)   |
| onkeyup     | JavaScript event handler executed when this element has focus and a key is released. (RT EXPR)   |
| onmousedown | JavaScript event handler executed when this element is under the mouse pointer and a mouse button is depressed. (RT EXPR)  |
| onmousemove | JavaScript event handler executed when this element is under the mouse pointer and the pointer is moved. (RT EXPR)   |
| onmouseout  | JavaScript event handler executed when this element was under the mouse pointer but the pointer was moved outside the element. (RT EXPR)   |
| onmouseover | JavaScript event handler executed when this element was not under the mouse pointer but the pointer is moved inside the element. (RT EXPR)   |
| onmouseup   | JavaScript event handler executed when this element is under the mouse pointer and a mouse button is released. (RT EXPR)   |
| property    | Name of this input field, and the name of the corresponding bean property if value is not specified. The corresponding bean property (if any) must be of type String. (REQUIRED) (RT EXPR) |
| readonly    | Set to <code>true</code> if this input field should be read only. (RT EXPR)  |
| size        | Number of character positions to allocate. [Browser default] (RT EXPR)   |
| style       | CSS styles to be applied to this HTML element. (RT EXPR)   |
| styleClass  | CSS stylesheet class to be applied to this HTML element (renders a "class" attribute). (RT EXPR)   |
| styleId     | Identifier to be assigned to this HTML element (renders an "id" attribute). (RT EXPR)  |
| tabindex    | The tab order (ascending positive integers) for this element. (RT EXPR)  |
| title       | The advisory title for this element.<br>(RT EXPR)  |
| titleKey    | The message resources key for the advisory title for this element.<br>(RT EXPR)  |
| value       | Value to which this field should be initialized. [Use the corresponding bean property value] (RT EXPR)   |

[Back to top](#)

## textarea - Render A Textarea

Render a textarea element. This tag is only valid when nested inside a form tag body.

| Attribute Name | Description  |
|----------------|--|
| accesskey      | The keyboard character used to move focus immediately to this element. (RT EXPR)   |
| alt            | The alternate text for this element.<br>(RT EXPR)  |
| altKey         | The message resources key of the alternate text for this element.<br>(RT EXPR)   |
| cols           | The number of columns to display. (RT EXPR)  |
| disabled       | Set to <code>true</code> if this input field should be disabled. (RT EXPR)   |
| indexed        | Valid only inside of <code>logic:iterate</code> tag. If <code>true</code> then name of the html tag will be rendered as " <code>id[34].propertyName</code> ". Number in brackets will be generated for every iteration and taken from ancestor <code>logic:iterate</code> tag. (RT EXPR) |
| name           | The attribute name of the bean whose properties are consulted when rendering the current value of this input field. If not specified, the bean associated with the form tag we are nested within is utilized. (RT EXPR)  |
| onblur         | JavaScript event handler executed when this element loses input focus. (RT EXPR)   |
| onchange       | JavaScript event handler executed when this element loses input focus and its value has changed. (RT EXPR)   |
| onclick        | JavaScript event handler executed when this element receives a mouse click. (RT EXPR)  |
| ondblclick     | JavaScript event handler executed when this element receives a mouse double click. (RT EXPR)   |
| onfocus        | JavaScript event handler executed when this element receives input focus. (RT EXPR)  |
| onkeydown      | JavaScript event handler executed when this element has focus and a key is depressed. (RT EXPR)  |
| onkeypress     | JavaScript event handler executed when this element has focus and a key is depressed and released. (RT EXPR)   |
| onkeyup        | JavaScript event handler executed when this element has focus and a key is released. (RT EXPR)   |
| onmousedown    | JavaScript event handler executed when this element is under the mouse pointer and a mouse button is depressed. (RT EXPR)  |
| onmousemove    | JavaScript event handler executed when this element is under the mouse pointer and the pointer is moved. (RT EXPR)   |
| onmouseout     | JavaScript event handler executed when this element was under the mouse pointer but the pointer was moved outside the element. (RT EXPR)   |
| onmouseover    | JavaScript event handler executed when this element was not under the mouse pointer but the pointer is moved inside the element. (RT EXPR)   |
| onmouseup      | JavaScript event handler executed when this element is under the mouse pointer and a mouse button is released. (RT EXPR)   |

|            |  |
|------------|--|
| property   | Name of this input field, and the name of the corresponding bean property if value is not specified. The corresponding bean property (if any) must be of type String. (REQUIRED) (RT EXPR) |
| readonly   | Set to <code>true</code> if this input field should be read only. (RT EXPR)  |
| rows       | The number of rows to display. (RT EXPR)   |
| style      | CSS styles to be applied to this HTML element. (RT EXPR)   |
| styleClass | CSS stylesheet class to be applied to this HTML element (renders a "class" attribute). (RT EXPR)   |
| styleId    | Identifier to be assigned to this HTML element (renders an "id" attribute). (RT EXPR)  |
| tabindex   | The tab order (ascending positive integers) for this element. (RT EXPR)  |
| title      | The advisory title for this element.<br>(RT EXPR)  |
| titleKey   | The message resources key for the advisory title for this element.<br>(RT EXPR)  |
| value      | Value to which this field should be initialized. [Use the corresponding bean property value] (RT EXPR)   |

[Back to top](#)

### xhtml - Render HTML tags as XHTML

Using this tag in a page tells all other html taglib tags to render themselves as XHTML. This is useful when composing pages with JSP includes or Tiles. This tag has the same effect as using `<html:html xhtml="true">`.

| Attribute Name | Description |
|----------------|-------------|
|----------------|-------------|

[Back to top](#)

# Package org.apache.struts.taglib.logic

The "struts-logic" tag library contains tags that are useful in managing conditional generation of output text, looping over object collections for repetitive generation of output text, and application flow management.

See: [Description](#)

| Class Summary                         |  |
|---------------------------------------|--|
| <a href="#">CompareTagBase</a>        | Abstract base class for comparison tags.   |
| <a href="#">ConditionalTagBase</a>    | Abstract base class for the various conditional evaluation tags.   |
| <a href="#">EmptyTag</a>              | Evalute the nested body content of this tag if the specified value is empty for this request.  |
| <a href="#">EqualTag</a>              | Evaluate the nested body content of this tag if the specified variable and value are equal.  |
| <a href="#">ForwardTag</a>            | Perform a forward or redirect to a page that is looked up in the configuration information associated with our application.  |
| <a href="#">GreaterEqualTag</a>       | Evaluate the nested body content of this tag if the specified variable is greater than or equal to the specified value.  |
| <a href="#">GreaterThanTag</a>        | Evaluate the nested body content of this tag if the specified variable is greater than the specified value.  |
| <a href="#">IterateTag</a>            | Custom tag that iterates the elements of a collection, which can be either an attribute or the property of an attribute.   |
| <a href="#">IterateTei</a>            | Implementation of TagExtraInfo for the <b>iterate</b> tag, identifying the scripting object(s) to be made visible.   |
| <a href="#">LessEqualTag</a>          | Evaluate the nested body content of this tag if the specified variable is less than or equal to the specified value.   |
| <a href="#">LessThanTag</a>           | Evaluate the nested body content of this tag if the specified variable is less than the specified value.   |
| <a href="#">MatchTag</a>              | Evalute the nested body content of this tag if the specified value is a substring of the specified variable.   |
| <a href="#">MessagesNotPresentTag</a> | Evalute the nested body content of this tag if the specified value is not present for this request.  |
| <a href="#">MessagesPresentTag</a>    | Evalute to true if an ActionMessages class or a class that can be converted to an ActionMessages class is in request scope under the specified key and there is at least one message in the class or for the property specified. |
| <a href="#">NotEmptyTag</a>           | Evalute the nested body content of this tag if the specified value is not empty for this request.  |
| <a href="#">NotEqualTag</a>           | Evaluate the nested body content of this tag if the specified variable and value are not equal.  |



|                               |   |
|-------------------------------|---|
| <a href="#">NotMatchTag</a>   | Evaluate the nested body content of this tag if the specified value is not a substring of the specified variable. |
| <a href="#">NotPresentTag</a> | Evaluate the nested body content of this tag if the specified value is not present for this request.              |
| <a href="#">PresentTag</a>    | Evaluate the nested body content of this tag if the specified value is present for this request.                  |
| <a href="#">RedirectTag</a>   | Generate a URL-encoded redirect to the specified URI.   |

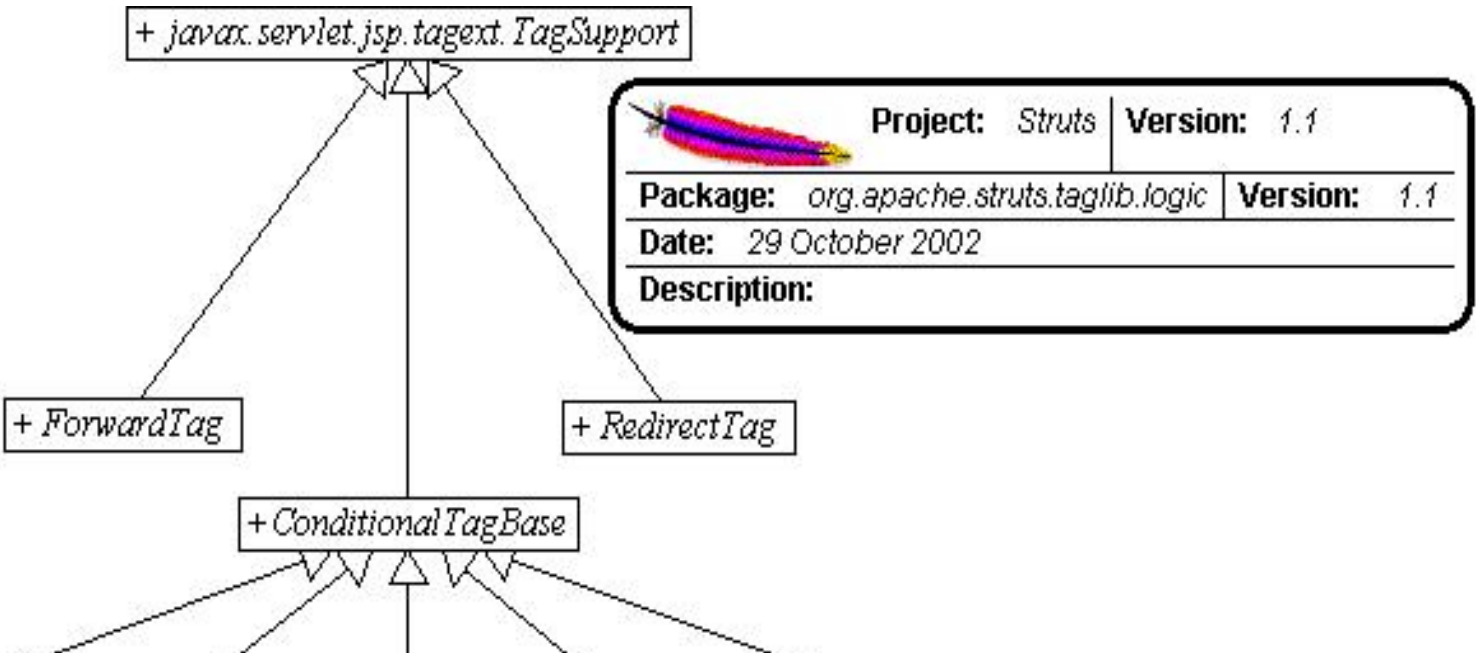
## Package org.apache.struts.taglib.logic Description

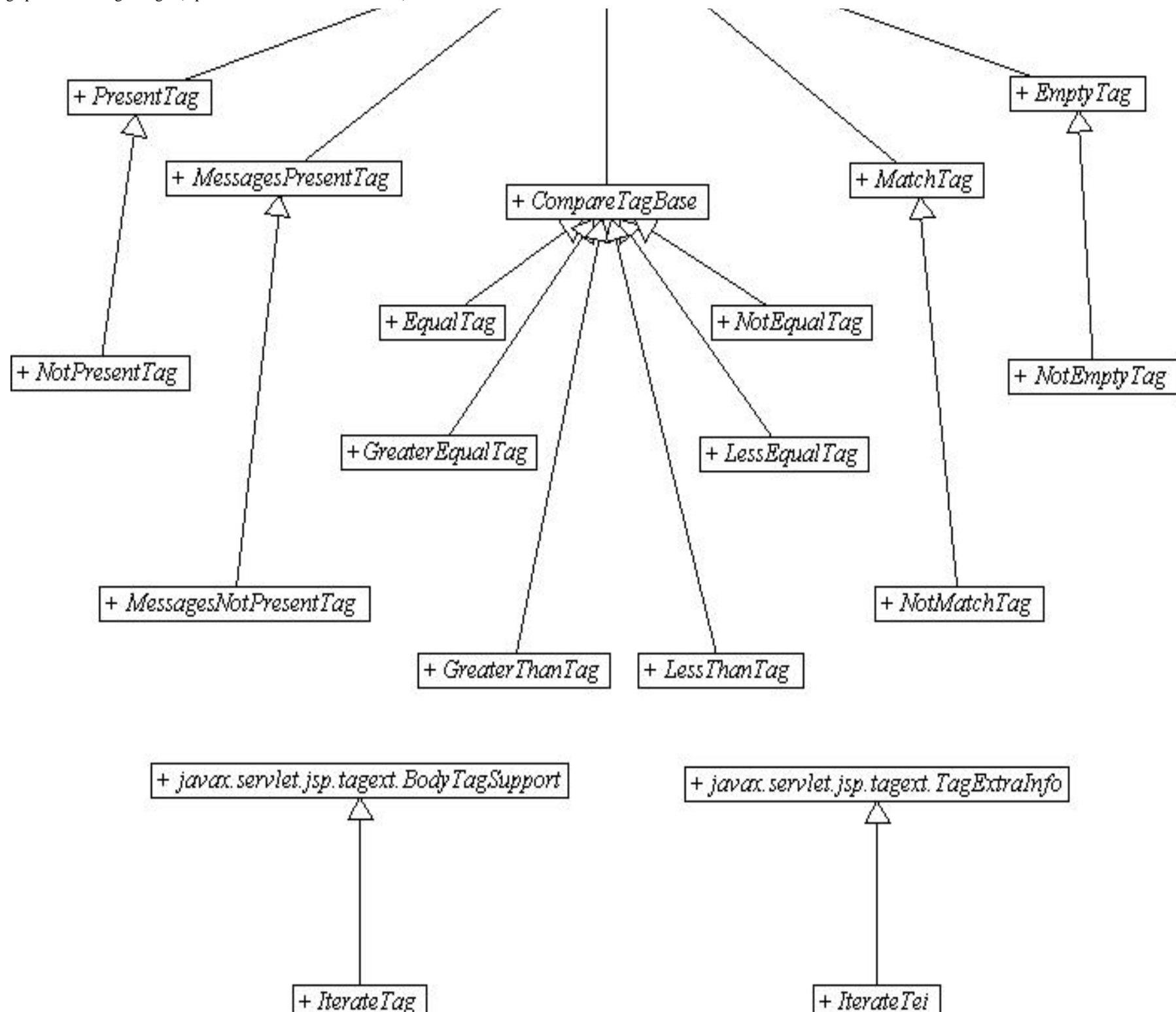
The "struts-logic" tag library contains tags that are useful in managing conditional generation of output text, looping over object collections for repetitive generation of output text, and application flow management.

[\[Introduction\]](#) [\[Logic Functionality\]](#) [\[Logic Properties\]](#) [\[Logic Examples\]](#)

### Introduction

The Logic library supplies tags that are useful for manipulating presentation logic without the use of scriptlets.





## Logic Tag Functionality

The functionality of the logic tags can be divided into four groups:

**Value Comparisons** - The purpose of these tags is to print out the body of the tag if the comparison evaluates to *true*.

- [equal](#), [notEqual](#)
- [greaterEqual](#) , [lessEqual](#)
- [greaterThan](#) , [lessThan](#)

**Substring Matching** - The purpose of these tags is to match substrings inside of other Strings

- [match](#) , [notMatch](#)

**Presentation Location** - The purpose of these tags is to change the location of the presentation page

- [forward](#)
- [redirect](#)

**Collection Utilities** -The purpose of these tags is to handle Collections

- [iterate](#)

## Logic Tag Properties

Each of the four groups of logic tags have a common set of attributes associated with them. :

**Value Comparisons** (equal, notEqual, greaterEqual, lessEqual, greaterThan, lessThan, present, notPresent)

Each of the value comparison tags takes a *value* and compares it to the value of a comparison attribute. If the value given can be successfully converted to a *float* or *double*, then a number comparison is performed on the value given and the value of the comparison attribute. Otherwise a String comparison is performed. You have to specify one of the comparison attributes: *cookie*, *header*, *parameter* , *property* or *name*. For each of the examples, the tag "*someComparisonTag*" can be replaced by any of the value comparison tags.

- value - the value to which this tag is going to compare, used in conjunction with one of the comparison attributes: *cookie*, *header*, *parameter*, and

*property* and/or *name*.

- cookie - the name of the cookie to compare to *value*
  - example:

```
<logic:someComparisonTag value="someUserName" cookie="userName">
    User Logged In
</logic:someComparisonTag>
```

- header - the name of the HTTP header to compare to *value*
  - example:

```
<logic:someComparisonTag value="en_US" header="Accept-Language">
    Welcome English-speaking User*
</logic:someComparisonTag>
```

- \*Note: See the section in the user's guide on [Internationalized Messages](#) to do things like this better.

- name - the variable to be compared to *value* is the JSP bean specified by this attribute, if property is not specified, or the value of the specified property of this bean, if property is specified.
  - example:

```
<%
    String testString = "pantalones";
    pageContext.setAttribute("testString", testString, PageContext.PAGE_SCOPE);
%>
<jsp:useBean id="testString" scope="page" type="java.lang.String" />
<logic:someComparisonTag name="testString" value="pantalones">
    Usted tiene pantalones!
</logic:someComparisonTag>
```

- parameter - the name of the request parameter to compare to *value*
  - example:

```
<logic:someComparisonTag value="" parameter="username">
    Error: a username must be specified
</logic:someComparisonTag>
```

- property - the variable to be compared with *value* is the property (of the bean specified by the name attribute) specified by this attribute. The property reference can be simple, nested, and/or indexed. *property* is used in conjunction with *name* to specify a property in the bean specified by *name*. For the type of syntax used for property, see the users guide on the Bean Tags.
- scope - the bean scope within which to search for the bean named by the name property, or "any scope" if not specified. Possible values are "page", "request", "session", "application", or "any scope"

## Substring Matching (match, notMatch)

The substring matching tags take all the same arguments as the value comparison tags. You compare the String specified by *value* to any of the comparison values you give it, specified by *cookie*, *header*, *parameter*, *property* or *name*. Note that in the examples, *matchTag* corresponds either the *match* or *notMatch* tag. Matching tags also have an additional *location* attribute added:

- location - has two possible values, "start" and "end". If "start", the substring is attempted to be matched at the beginning of the String, if "end", then the substring is attempted to be matched to the end of the String
  - example:

```
<logic:matchTag parameter="action" value="processLogin" location="start">
  Processing Login....
</logic:matchTag>
```

In this example, a request parameter "action" was compared to see if its value started with the String "processLogin". In this case, *matchTag* would have to be `<logic:match>`.

## Presentation Location (forward, redirect)

The *redirect* tag is responsible for sending a re-direct to the client's browser, complete with URL-rewriting if it's supported by the container. Its attributes are consistent with the Struts HTML [link](#) tag. The base URL is calculated based on which of the following attributes you specify (you must specify exactly one of them):

- forward - Use the value of this attribute as the name of a global ActionForward to be looked up, and use the context-relative URI found there.
- href - Use the value of this attribute unchanged.
- page - Use the value of this attribute as a context-relative URI, and generate a server-relative URI by including the context path.

The *forward* tag is responsible for either redirecting or forwarding to a specified global action forward. To define a global ActionForward, see The [Action Mappings Configuration File](#) . You can specify whether the forward re-directs or forwards when executed in the config file. The forward tag has one attribute:

- name - The logical name of the ActionForward to use

## Collection Utilities (iterate)

The *iterate* tag is responsible for executing its body content once for every element inside of the specified Collection. There is one required attribute:

- id - The name of a page scope JSP bean that will contain the current element of the collection on each iteration

The other attributes allow for more flexibility on which Collection to iterate and how to do it:

- collection - a runtime expression that evaluates to a Collection to be iterated
  - example:

```
<%
    java.util.Vector vector = new java.util.Vector();
    vector.add(new Integer(12));
    vector.add(new Integer(5));
%>

<logic:iterate id="myCollectionElement" collection="<%= vector %>">
    Do something with myCollectionElement
</logic:iterate>
```

- length - The maximum number of entries (from the underlying collection) to be iterated through on this page. This can be either an integer that directly expresses the desired value, or the name of a JSP bean (in any scope) of type java.lang.Integer that defines the desired value. If not present, there will be no limit on the number of iterations performed
- name - The name of the JSP bean containing the collection to be iterated (if property is not specified), or the JSP bean whose property getter returns the collection to be iterated (if property is specified).
  - example:

```
<%

    java.util.ArrayList list = new java.util.ArrayList();
    list.add("First");
    list.add("Second");
    list.add("Third");
    list.add("Fourth");
    list.add("Fifth");
    pageContext.setAttribute("list", list, PageContext.PAGE_SCOPE);
%>

<logic:iterate id="myCollectionElement" name="list">
    Do something with myCollectionElement
</logic:iterate>
```

- offset - The zero-relative index of the starting point at which entries from the underlying collection will be iterated through. This can be either an integer that directly expresses the desired value, or the name of a JSP bean (in any scope) of type java.lang.Integer that defines the desired value. If not present, zero is assumed (meaning that the collection will be iterated from the beginning).
- property - Name of the property, of the JSP bean specified by name, whose getter returns the collection to be iterated. See the user's guide for the bean tag library for the syntax of the property attribute
- scope - The bean scope within which to search for the bean named by the name property, or "any scope" if not specified. Possible values are "page", "request", "session", "application", or "any scope"
- type - Fully qualified Java class name of the element to be exposed through the JSP bean named from the id attribute. If not present, no type

conversions will be performed. NOTE: The actual elements of the collection must be assignment-compatible with this class, or a request time `ClassCastException` will occur.

- o example:

```
<%      java.util.ArrayList list = new java.util.ArrayList();
list.add("First");
list.add("Second");
list.add("Third");
list.add("Fourth");
list.add("Fifth");
      pageContext.setAttribute("list", list, PageContext.PAGE_SCOPE);
%>
<logic:iterate id="myCollectionElement" name="list" type="java.lang.String">
  Do something with myCollectionElement
</logic:iterate>
```

## Logic Examples

### Value Comparisons

#### Logic Equivalence Tags (equal, notEqual, greaterEqual, lessEqual, lessThan, greaterThan)

You can compare these tags to the "=", "!=" , ">=", "<=", "<", and ">" logic operators in most languages. Their usage is fairly straightforward for numbers. For an example, we'll create a small "Guess That Number" game that uses request parameters from a form input to play. The number will be hardcoded as "7", because this is just an example. Note that this is actually putting application logic inside of jsp pages, and isn't the recommended development method for Struts. It's just an easy way to show how these tags are used:

The first step is to develop the form that will call on the processing jsp page. This form will use the "GET" method so that you can see the request parameter in the URL. The POST method can also be used with no problem or changes.

[numberGuess.jsp]

```
<form action="numberProcess.jsp" method="GET">
Please Enter a Number From 1-10: <input type="text" name="number" /><br />
  <center>
    <input type="submit" name="Guess Number" />
  </center>
</form>
```

The next step is to create the processing page. It uses the struts-logic taglib. For information on how to set this tag library up in your application to use, see [The Web Application Deployment Descriptor](http://jakarta.apache.org/struts/api/org/apache/struts/taglib/logic/package-summary.html)

[numberProcess.jsp]

```

<%@ page language="java" %>
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>

<!-- Is the number guess right? -->
<logic:equal parameter="number" value="7">
    You guessed right! You win a high speed blender!
</logic:equal>

<!-- If the number guessed was wrong -->
<logic:notEqual parameter="number" value="7">
    <!-- Less Than -->
    <logic:lessThan parameter="number" value="7">
        A little higher...
    </logic:lessThan>
    <!-- Greater Than -->
    <logic:greaterThan parameter="number" value="7">
        A little lower...
    </logic:greaterThan>
</logic:notEqual>

```

Basically, the numberProcess.jsp page uses the equal tag to check if the guess is 7, and if so, prints out a congratulatory message. If the number isn't equal, specified by the use of the <logic:notEqual> tag, it uses the greaterThan and lessThan tags to check if the number is higher or lower than 7, and prints out a hint. As said before, this is a horribly designed small application, with no validity checks on the number input, but shows the basic usage of the logic equal tags

For String comparisons, the equal tags use the java.lang.String.compareTo() method. See the javadocs on the compareTo() method for more information, located [here](#) .

### Match and Present Tags (match, notMatch, present, notPresent)

You use the match tags in conjunction with the present tags in order to do substring matches. For an example using this we'll use headers, specifically the "Referer" header. The HTTP referer header gives the URL of the document that refers to the requested URL. We'll use this to check if the user is coming from a link specified by a [Google](#) search, and offer a personalized greeting, frightening users that find our site through the search engine with our amazing intimate knowledge of their browsing habits:

[sneaky.jsp]

```

<%@ page language="java" %>

```



```

<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>
<!-- Check to see if the "Referer" header is present -->
<logic:present header="Referer">
  <logic:match header="Referer" value="google.com">
    I see you found our site through Google... interesting.
  </logic:match>
  <logic:notMatch header="Referer" value="google.com">
    Welcome to the site, we're secretly logging what site you came from,
    because we're shady...
  </logic:notMatch>
</logic:present>
<!-- If the header is not present -->
<logic:notPresent header="Referer">
  Hi, welcome to our site. Please fill out our
  <a href="nonExistantForm.jsp">Form</a> and
  tell us where you're coming from.
</logic:notPresent>

```

Note: Another interesting usage of these tags and headers would be to use the "User-Agent" header to display browser-specific javascript.

## Collection Utilities (iterate)

For an example of using the `<logic:iterate>` tag, we'll use one of the previous examples given, in it's entirety. This example uses the `<bean:write>` tag from the Bean Tag Library, see the User's Guide on the bean tag library for more information on it's usage:

[iterate.jsp]

```

<%@ page language="java" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>

<%
java.util.ArrayList list = new java.util.ArrayList();
  list.add("First");
  list.add("Second");
  list.add("Third");
  list.add("Fourth");
  list.add("Fifth");
  pageContext.setAttribute("list", list, PageContext.PAGE_SCOPE);
%>
<logic:iterate id="myCollectionElement" name="list">
  Element Value: <bean:write name="myCollectionElement" /><br />

```

```
</logic:iterate>
```

[Overview](#) **Package** [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV PACKAGE](#) [NEXT PACKAGE](#) [FRAMES](#) [NO FRAMES](#) [All Classes](#)



## User Guide

[Table of Contents](#)

[Preface](#)

[Introduction](#)

[Model Components](#)

[View Components](#)

[Controller Components](#)

[Configuration](#)

[Release Notes](#)

[Installation](#)

## Developer Guides

[Bean Tags](#)

[HTML Tags](#)

[Logic Tags](#)

[Nested Tags](#)

[Template Tags](#)

[Tiles Tags](#)

[Utilities](#)

[Validator](#)

## Quick Links

[Welcome](#)

[News and Status](#)

[Resources](#)

[User and Developer Guides \\*](#)

[FAQs and HowTos](#)

## Struts Logic Tags

This tag library contains tags that are useful in managing conditional generation of output text, looping over object collections for repetitive generation of output text, and application flow management.

For tags that do value comparisons (`equal`, `greaterEqual`, `greaterThan`, `lessEqual`, `lessThan`, `notEqual`), the following rules apply:

- The specified value is examined. If it can be converted successfully to a double or a long, it is assumed that the ultimate comparison will be numeric (either floating point or integer). Otherwise, a String comparison will be performed.
- The variable to be compared to is retrieved, based on the selector attribute(s) (`cookie`, `header`, `name`, `parameter`, `property`) present on this tag. It will be converted to the appropriate type for the comparison, as determined above.
- If the specified variable or property returns null, it will be coerced to a zero-length string before the comparison occurs.
- The specific comparison for this tag will be performed, and the nested body content of this tag will be evaluated if the comparison returns a true result.

For tags that do substring matching (`match`, `notMatch`), the following rules apply:

- The specified variable is retrieved, based on the selector attribute(s) (`cookie`, `header`, `name`, `parameter`, `property`) present on this tag. The variable is converted to a String, if necessary.
- A request time exception will be thrown if the specified variable cannot be retrieved, or has a null value.
- The specified value is checked for existence as a substring of the variable, in the position specified by the `location` attribute, as follows: at the beginning (if `location` is set to `start`), at the end (if `location` is set to `end`), or anywhere (if `location` is not specified).

Many of the tags in this tag library will throw a `JspException` at runtime when they are utilized incorrectly (such as when you specify an invalid combination of tag attributes). JSP allows you to declare an "error page" in the `<%@ page %>` directive. If you wish to process the actual exception that caused the problem, it is passed to the error page as a request attribute under key `org.apache.struts.action.EXCEPTION`.

| Tag Name                                  | Description   |
|---|---|
| <a href="#"><u>empty</u></a>              | Evaluate the nested body content of this tag if the requested variable is either null or an empty string.   |
| <a href="#"><u>equal</u></a>              | Evaluate the nested body content of this tag if the requested variable is equal to the specified value.   |
| <a href="#"><u>forward</u></a>            | Forward control to the page specified by the specified ActionForward entry.   |
| <a href="#"><u>greaterEqual</u></a>       | Evaluate the nested body content of this tag if the requested variable is greater than or equal to the specified value.   |
| <a href="#"><u>greaterThan</u></a>        | Evaluate the nested body content of this tag if the requested variable is greater than the specified value.   |
| <a href="#"><u>iterate</u></a>            | Repeat the nested body content of this tag over a specified collection.   |
| <a href="#"><u>lessEqual</u></a>          | Evaluate the nested body content of this tag if the requested variable is greater than or equal to the specified value.   |
| <a href="#"><u>lessThan</u></a>           | Evaluate the nested body content of this tag if the requested variable is less than the specified value.  |
| <a href="#"><u>match</u></a>              | Evaluate the nested body content of this tag if the specified value is an appropriate substring of the requested variable.  |
| <a href="#"><u>messagesNotPresent</u></a> | Generate the nested body content of this tag if the specified message is not present in this request.   |
| <a href="#"><u>messagesPresent</u></a>    | Generate the nested body content of this tag if the specified message is present in this request.   |
| <a href="#"><u>notEmpty</u></a>           | Evaluate the nested body content of this tag if the requested variable is neither null, nor an empty string, nor an empty java.util.Collection (tested by the .isEmpty() method on the java.util.Collection interface). |
| <a href="#"><u>notEqual</u></a>           | Evaluate the nested body content of this tag if the requested variable is not equal to the specified value.   |
| <a href="#"><u>notMatch</u></a>           | Evaluate the nested body content of this tag if the specified value is not an appropriate substring of the requested variable.  |
| <a href="#"><u>notPresent</u></a>         | Generate the nested body content of this tag if the specified value is not present in this request.   |
| <a href="#"><u>present</u></a>            | Generate the nested body content of this tag if the specified value is present in this request.   |
| <a href="#"><u>redirect</u></a>           | Render an HTTP Redirect   |

**empty** - Evaluate the nested body content of this tag if the requested variable is either null or an empty string.

Since: Struts 1.1

This tag evaluates its nested body content only if the specified value is either absent (i.e. `null`), an empty string (i.e. a `java.lang.String` with a length of zero), or an empty `java.util.Collection` or `java.util.Map` (tested by the `isEmpty()` method on the respective interface).

| Attribute Name | Description  |
|----------------|--|
| name           | The variable to be compared is the JSP bean specified by this attribute, if <code>property</code> is not specified, or the value of the specified property of this bean, if <code>property</code> is specified.<br><br>(RT EXPR) |
| property       | The variable to be compared is the property (of the bean specified by the <code>name</code> attribute) specified by this attribute. The property reference can be simple, nested, and/or indexed.<br><br>(RT EXPR)               |
| scope          | The bean scope within which to search for the bean named by the <code>name</code> property, or "any scope" if not specified.<br><br>(RT EXPR)  |

[Back to top](#)

**equal** - Evaluate the nested body content of this tag if the requested variable is equal to the specified value.

Compares the variable specified by one of the selector attributes against the specified constant value. The nested body content of this tag is evaluated if the variable and value are **equal**.

| Attribute Name | Description  |
|----------------|--|
| cookie         | The variable to be compared is the value of the cookie whose name is specified by this attribute.<br><br>(RT EXPR)   |
| header         | The variable to be compared is the value of the header whose name is specified by this attribute. The name match is performed in a case insensitive manner.<br><br>(RT EXPR)   |
| name           | The variable to be compared is the JSP bean specified by this attribute, if <code>property</code> is not specified, or the value of the specified property of this bean, if <code>property</code> is specified.<br><br>(RT EXPR) |

|           |   |
|-----------|---|
| parameter | The variable to be compared is the first, or only, value of the request parameter specified by this attribute.<br><br>(RT EXPR)   |
| property  | The variable to be compared is the property (of the bean specified by the name attribute) specified by this attribute. The property reference can be simple, nested, and/or indexed.<br><br>(RT EXPR) |
| scope     | The bean scope within which to search for the bean named by the name property, or "any scope" if not specified.<br><br>(RT EXPR)  |
| value     | The constant value to which the variable, specified by other attribute(s) of this tag, will be compared.<br><br>(REQUIRED) (RT EXPR)  |

[Back to top](#)

**forward** - Forward control to the page specified by the specified ActionForward entry.

Performs a `PageContext.forward()` or `HttpServletResponse.sendRedirect()` call for the global ActionForward entry for the specified name. URL rewriting will occur automatically if a redirect is performed.

| Attribute Name | Description  |
|----------------|--|
| name           | The logical name of the global ActionForward entry that identifies the destination, and forwarding approach, to be used.<br><br>(REQUIRED) (RT EXPR) |

[Back to top](#)

**greaterEqual** - Evaluate the nested body content of this tag if the requested variable is greater than or equal to the specified value.

Compares the variable specified by one of the selector attributes against the specified constant value. The nested body content of this tag is evaluated if the variable is **greater than or equal** to the value.

| Attribute Name | Description  |
|----------------|--|
| cookie         | The variable to be compared is the value of the cookie whose name is specified by this attribute.<br><br>(RT EXPR)   |
| header         | The variable to be compared is the value of the header whose name is specified by this attribute. The name match is performed in a case insensitive manner.<br><br>(RT EXPR)   |
| name           | The variable to be compared is the JSP bean specified by this attribute, if <code>property</code> is not specified, or the value of the specified property of this bean, if <code>property</code> is specified.<br><br>(RT EXPR) |
| parameter      | The variable to be compared is the first, or only, value of the request parameter specified by this attribute.<br><br>(RT EXPR)  |
| property       | The variable to be compared is the property (of the bean specified by the name attribute) specified by this attribute. The property reference can be simple, nested, and/or indexed.<br><br>(RT EXPR)                            |
| scope          | The bean scope within which to search for the bean named by the name property, or "any scope" if not specified.<br><br>(RT EXPR)   |
| value          | The constant value to which the variable, specified by other attribute(s) of this tag, will be compared.<br><br>(REQUIRED) (RT EXPR)   |

[Back to top](#)

**greaterThan** - Evaluate the nested body content of this tag if the requested variable is greater than the specified value.

Compares the variable specified by one of the selector attributes against the specified constant value. The nested body content of this tag is evaluated if the variable is **greater than** the value.

| Attribute Name | Description  |
|----------------|--|
| cookie         | The variable to be compared is the value of the cookie whose name is specified by this attribute.<br><br>(RT EXPR)   |
| header         | The variable to be compared is the value of the header whose name is specified by this attribute. The name match is performed in a case insensitive manner.<br><br>(RT EXPR)   |
| name           | The variable to be compared is the JSP bean specified by this attribute, if <code>property</code> is not specified, or the value of the specified property of this bean, if <code>property</code> is specified.<br><br>(RT EXPR) |
| parameter      | The variable to be compared is the first, or only, value of the request parameter specified by this attribute.<br><br>(RT EXPR)  |
| property       | The variable to be compared is the property (of the bean specified by the name attribute) specified by this attribute. The property reference can be simple, nested, or indexed.<br><br>(RT EXPR)                                |
| scope          | The bean scope within which to search for the bean named by the name property, or "any scope" if not specified.<br><br>(RT EXPR)   |
| value          | The constant value to which the variable, specified by other attribute(s) of this tag, will be compared.<br><br>(REQUIRED) (RT EXPR)   |

[Back to top](#)

**iterate** - Repeat the nested body content of this tag over a specified collection.

Repeats the nested body content of this tag once for every element of the specified collection, which must be an `Iterator`, a `Collection`, a `Map` (whose values are to be iterated over), or an array. The collection to be iterated over must be specified in one of the following ways:

- As a runtime expression specified as the value of the `collection` attribute.
- As a JSP bean specified by the `name` attribute.
- As the property, specified by the `property`, of the JSP bean specified by the `name` attribute.

The collection to be iterated over **MUST** conform to one of the following requirements in order for iteration to be successful:



- An array of Java objects or primitives.
- An implementation of `java.util.Collection`, including `ArrayList` and `Vector`.
- An implementation of `java.util.Enumeration`.
- An implementation of `java.util.Iterator`.
- An implementation of `java.util.Map`, including `HashMap`, `Hashtable`, and `TreeMap`. **NOTE** - See below for additional information about accessing Maps.

Normally, each object exposed by the `iterate` tag is an element of the underlying collection you are iterating over. However, if you iterate over a `Map`, the exposed object is of type `Map.Entry` that has two properties:

- `key` - The key under which this item is stored in the underlying `Map`.
- `value` - The value that corresponds to this key.

So, if you wish to iterate over the values of a `Hashtable`, you would implement code like the following:

```
<logic:iterate id="element" name="myhashtable">
Next element is <bean:write name="element"
property="value" />
</logic:iterate>
```

If the collection you are iterating over can contain `null` values, the loop will still be performed but no page scope attribute (named by the `id` attribute) will be created for that loop iteration. You can use the `<logic:present>` and `<logic:notPresent>` tags to test for this case.

| Attribute Name | Description   |
|----------------|---|
| collection     | A runtime expression that evaluates to a collection (conforming to the requirements listed above) to be iterated over.<br><br>(RT EXPR)   |
| id             | The name of a page scope JSP bean that will contain the current element of the collection on each iteration, if it is not <code>null</code> .<br><br>(REQUIRED) (RT EXPR)   |
| indexId        | The name of a page scope JSP bean that will contain the current index of the collection on each iteration.<br><br>(RT EXPR)   |
| length         | The maximum number of entries (from the underlying collection) to be iterated through on this page. This can be either an integer that directly expresses the desired value, or the name of a JSP bean (in any scope) of type <code>java.lang.Integer</code> that defines the desired value. If not present, there will be no limit on the number of iterations performed.<br><br>(RT EXPR) |

|          |  |
|----------|--|
| name     | The name of the JSP bean containing the collection to be iterated (if <code>property</code> is not specified), or the JSP bean whose property getter returns the collection to be iterated (if <code>property</code> is specified).<br><br>(RT EXPR)   |
| offset   | The zero-relative index of the starting point at which entries from the underlying collection will be iterated through. This can be either an integer that directly expresses the desired value, or the name of a JSP bean (in any scope) of type <code>java.lang.Integer</code> that defines the desired value. If not present, zero is assumed (meaning that the collection will be iterated from the beginning).<br><br>(RT EXPR) |
| property | Name of the property, of the JSP bean specified by <code>name</code> , whose getter returns the collection to be iterated.<br><br>(RT EXPR)  |
| scope    | The bean scope within which to search for the bean named by the <code>name</code> property, or "any scope" if not specified.<br><br>(RT EXPR)  |
| type     | Fully qualified Java class name of the element to be exposed through the JSP bean named from the <code>id</code> attribute. If not present, no type conversions will be performed. NOTE: The actual elements of the collection must be assignment-compatible with this class, or a request time <code>ClassCastException</code> will occur.<br><br>(RT EXPR)   |

[Back to top](#)

**lessEqual** - Evaluate the nested body content of this tag if the requested variable is greater than or equal to the specified value.

Compares the variable specified by one of the selector attributes against the specified constant value. The nested body content of this tag is evaluated if the variable is **less than or equal** to the value.

| Attribute Name | Description  |
|----------------|--|
| cookie         | The variable to be compared is the value of the cookie whose name is specified by this attribute.<br><br>(RT EXPR)   |
| header         | The variable to be compared is the value of the header whose name is specified by this attribute. The name match is performed in a case insensitive manner.<br><br>(RT EXPR) |

|           |  |
|-----------|--|
| name      | The variable to be compared is the JSP bean specified by this attribute, if <code>property</code> is not specified, or the value of the specified property of this bean, if <code>property</code> is specified.<br><br>(RT EXPR) |
| parameter | The variable to be compared is the first, or only, value of the request parameter specified by this attribute.<br><br>(RT EXPR)  |
| property  | The variable to be compared is the property (of the bean specified by the name attribute) specified by this attribute. The property reference can be simple, nested, or indexed.<br><br>(RT EXPR)                                |
| scope     | The bean scope within which to search for the bean named by the name property, or "any scope" if not specified.<br><br>(RT EXPR)   |
| value     | The constant value to which the variable, specified by other attribute(s) of this tag, will be compared.<br><br>(REQUIRED) (RT EXPR)   |

[Back to top](#)

**lessThan** - Evaluate the nested body content of this tag if the requested variable is less than the specified value.

Compares the variable specified by one of the selector attributes against the specified constant value. The nested body content of this tag is evaluated if the variable is **less than** the value.

| Attribute Name | Description  |
|----------------|--|
| cookie         | The variable to be compared is the value of the cookie whose name is specified by this attribute.<br><br>(RT EXPR)   |
| header         | The variable to be compared is the value of the header whose name is specified by this attribute. The name match is performed in a case insensitive manner.<br><br>(RT EXPR)   |
| name           | The variable to be compared is the JSP bean specified by this attribute, if <code>property</code> is not specified, or the value of the specified property of this bean, if <code>property</code> is specified.<br><br>(RT EXPR) |

|           |   |
|-----------|---|
| parameter | The variable to be compared is the first, or only, value of the request parameter specified by this attribute.<br><br>(RT EXPR)   |
| property  | The variable to be compared is the property (of the bean specified by the name attribute) specified by this attribute. The property reference can be simple, nested, and/or indexed.<br><br>(RT EXPR) |
| scope     | The bean scope within which to search for the bean named by the name property, or "any scope" if not specified.<br><br>(RT EXPR)  |
| value     | The constant value to which the variable, specified by other attribute(s) of this tag, will be compared.<br><br>(REQUIRED) (RT EXPR)  |

[Back to top](#)

**match** - Evaluate the nested body content of this tag if the specified value is an appropriate substring of the requested variable.

Matches the variable specified by one of the selector attributes (as a String) against the specified constant value. If the value is a substring (appropriately limited by the `location` attribute), the nested body content of this tag is evaluated.

| Attribute Name | Description   |
|----------------|---|
| cookie         | The variable to be matched is the value of the cookie whose name is specified by this attribute.<br><br>(RT EXPR)   |
| header         | The variable to be matched is the value of the header whose name is specified by this attribute. The name match is performed in a case insensitive manner.<br><br>(RT EXPR)   |
| location       | If not specified, a match between the variable and the value may occur at any position within the variable string. If specified, the match must occur at the specified location (either <code>start</code> or <code>end</code> ) of the variable string.<br><br>(RT EXPR) |
| name           | The variable to be matched is the JSP bean specified by this attribute, if <code>property</code> is not specified, or the value of the specified property of this bean, if <code>property</code> is specified.<br><br>(RT EXPR)   |

|           |  |
|-----------|--|
| parameter | The variable to be matched is the first, or only, value of the request parameter specified by this attribute.<br><br>(RT EXPR)   |
| property  | The variable to be matched is the property (of the bean specified by the name attribute) specified by this attribute. The property reference can be simple, nested, and/or indexed.<br><br>(RT EXPR) |
| scope     | The bean scope within which to search for the bean named by the name property, or "any scope" if not specified.<br><br>(RT EXPR)   |
| value     | The constant value which is checked for existence as a substring of the specified variable.<br><br>(REQUIRED) (RT EXPR)  |

[Back to top](#)

**messagesNotPresent** - Generate the nested body content of this tag if the specified message is not present in this request.

Since: Struts 1.1

Evaluates the nested body content of this tag if an `ActionMessages` object, `ActionErrors` object, a `String`, or a `String` array is not in request scope. If such a bean is not found, nothing will be rendered.

| Attribute Name | Description  |
|----------------|--|
| message        | By default the tag will retrieve the request scope bean it will iterate over from the <code>Action.ERROR_KEY</code> constant string, but if this attribute is set to 'true' the request scope bean will be retrieved from the <code>Action.MESSAGE_KEY</code> constant string. Also if this is set to 'true', any value assigned to the name attribute will be ignored.<br><br>(RT EXPR) |
| name           | The parameter key to retrieve the message from request scope.<br><br>(RT EXPR)   |
| property       | Name of the property for which messages should be retrieved. If not specified, all messages (regardless of property) are retrieved.<br><br>(RT EXPR)   |

[Back to top](#)

**messagesPresent** - Generate the nested body content of this tag if the specified message is present in this request.

Since: Struts 1.1

Evaluates the nested body content of this tag if an `ActionMessages` object, `ActionErrors` object, a `String`, or a `String` array is in request scope. If such a bean is not found, nothing will be rendered.

| Attribute Name | Description  |
|----------------|--|
| message        | By default the tag will retrieve the request scope bean it will iterate over from the <code>Action.ERROR_KEY</code> constant string, but if this attribute is set to 'true' the request scope bean will be retrieved from the <code>Action.MESSAGE_KEY</code> constant string. Also if this is set to 'true', any value assigned to the name attribute will be ignored.<br><br>(RT EXPR) |
| name           | The parameter key to retrieve the message from request scope.<br><br>(RT EXPR)   |
| property       | Name of the property for which messages should be retrieved. If not specified, all messages (regardless of property) are retrieved.<br><br>(RT EXPR)   |

[Back to top](#)

**notEmpty** - Evaluate the nested body content of this tag if the requested variable is neither null, nor an empty string, nor an empty `java.util.Collection` (tested by the `.isEmpty()` method on the `java.util.Collection` interface).

This tag evaluates its nested body content only if the specified value is present (i.e. not null) and is not an empty string (i.e. a `java.lang.String` with a length of zero).

| Attribute Name | Description  |
|----------------|--|
| name           | The variable to be compared is the JSP bean specified by this attribute, if <code>property</code> is not specified, or the value of the specified property of this bean, if <code>property</code> is specified.<br><br>(RT EXPR) |
| property       | The variable to be compared is the property (of the bean specified by the name attribute) specified by this attribute. The property reference can be simple, nested, and/or indexed.<br><br>(RT EXPR)                            |
| scope          | The bean scope within which to search for the bean named by the name property, or "any scope" if not specified.<br><br>(RT EXPR)   |

[Back to top](#)

**notEqual** - Evaluate the nested body content of this tag if the requested variable is not equal to the specified value.

Compares the variable specified by one of the selector attributes against the specified constant value. The nested body content of this tag is evaluated if the variable and value are **not equal**.

| Attribute Name | Description  |
|----------------|--|
| cookie         | The variable to be compared is the value of the cookie whose name is specified by this attribute.<br><br>(RT EXPR)   |
| header         | The variable to be compared is the value of the header whose name is specified by this attribute. The name match is performed in a case insensitive manner.<br><br>(RT EXPR)   |
| name           | The variable to be compared is the JSP bean specified by this attribute, if <code>property</code> is not specified, or the value of the specified property of this bean, if <code>property</code> is specified.<br><br>(RT EXPR) |
| parameter      | The variable to be compared is the first, or only, value of the request parameter specified by this attribute.<br><br>(RT EXPR)  |
| property       | The variable to be compared is the property (of the bean specified by the name attribute) specified by this attribute. The property reference can be simple, nested, and/or indexed.<br><br>(RT EXPR)                            |
| scope          | The bean scope within which to search for the bean named by the name property, or "any scope" if not specified.<br><br>(RT EXPR)   |
| value          | The constant value to which the variable, specified by other attribute(s) of this tag, will be compared.<br><br>(REQUIRED) (RT EXPR)   |

[Back to top](#)

**notMatch** - Evaluate the nested body content of this tag if the specified value is not an appropriate substring of the requested variable.

Matches the variable specified by one of the selector attributes (as a String) against the specified constant value. If the value is not a substring (appropriately limited by the `location` attribute), the nested body content of this tag is evaluated.

| Attribute Name | Description   |
|----------------|---|
| cookie         | The variable to be matched is the value of the cookie whose name is specified by this attribute.<br><br>(RT EXPR)   |
| header         | The variable to be matched is the value of the header whose name is specified by this attribute. The name match is performed in a case insensitive manner.<br><br>(RT EXPR)   |
| location       | If not specified, a match between the variable and the value may occur at any position within the variable string. If specified, the match must occur at the specified location (either <code>start</code> or <code>end</code> ) of the variable string.<br><br>(RT EXPR) |
| name           | The variable to be matched is the JSP bean specified by this attribute, if <code>property</code> is not specified, or the value of the specified property of this bean, if <code>property</code> is specified.<br><br>(RT EXPR)   |
| parameter      | The variable to be matched is the first, or only, value of the request parameter specified by this attribute.<br><br>(RT EXPR)  |
| property       | The variable to be matched is the property (of the bean specified by the <code>name</code> attribute) specified by this attribute. The property reference can be simple, nested, and/or indexed.<br><br>(RT EXPR)   |
| scope          | The bean scope within which to search for the bean named by the <code>name</code> property, or "any scope" if not specified.<br><br>(RT EXPR)   |
| value          | The constant value which is checked for existence as a substring of the specified variable.<br><br>(REQUIRED) (RT EXPR)   |

[Back to top](#)

**notPresent** - Generate the nested body content of this tag if the specified value is not present in this request.



Depending on which attribute is specified, this tag checks the current request, and evaluates the nested body content of this tag only if the specified value **is not** present. Only one of the attributes may be used in one occurrence of this tag, unless you use the `property` attribute, in which case the `name` attribute is also required.

| Attribute Name | Description  |
|----------------|--|
| cookie         | Checks for the existence of a cookie with the specified name.<br>(RT EXPR)   |
| header         | Checks for the existence of an HTTP header with the specified name. The name match is performed in a case insensitive manner.<br>(RT EXPR)   |
| name           | Checks for the existence of a JSP bean, in any scope, with the specified name. If <code>property</code> is also specified, checks for a non-null property value for the specified property.<br>(RT EXPR)   |
| parameter      | Checks for the existence of at least one occurrence of the specified request parameter on this request, even if the parameter value is a zero-length string.<br>(RT EXPR)  |
| property       | Checks for the existence of a non-null property value, returned by a property getter method on the JSP bean (in any scope) that is specified by the <code>name</code> attribute. Property references can be simple, nested, and/or indexed.<br>(RT EXPR) |
| role           | Checks whether the currently authenticated user (if any) has been associated with the specified security role.<br>(RT EXPR)  |
| scope          | The bean scope within which to search for the bean named by the <code>name</code> property, or "any scope" if not specified.<br>(RT EXPR)  |
| user           | Checks whether the currently authenticated user principal has the specified name.<br>(RT EXPR)   |

[Back to top](#)

**present** - Generate the nested body content of this tag if the specified value is present in this request.

Depending on which attribute is specified, this tag checks the current request, and evaluates the nested body content of this tag only if the specified value **is** present. Only one of the attributes may be used in one occurrence of this tag, unless you use the `property` attribute, in which case the `name` attribute is also required.

| Attribute Name | Description  |
|----------------|--|
| cookie         | Checks for the existence of a cookie with the specified name.<br><br>(RT EXPR)   |
| header         | Checks for the existence of an HTTP header with the specified name. The name match is performed in a case insensitive manner.<br><br>(RT EXPR)   |
| name           | Checks for the existence of a JSP bean, in any scope, with the specified name. If <code>property</code> is also specified, checks for a non-null property value for the specified property.<br><br>(RT EXPR)   |
| parameter      | Checks for the existence of at least one occurrence of the specified request parameter on this request, even if the parameter value is a zero-length string.<br><br>(RT EXPR)  |
| property       | Checks for the existence of a non-null property value, returned by a property getter method on the JSP bean (in any scope) that is specified by the <code>name</code> attribute. Property references can be simple, nested, and/or indexed.<br><br>(RT EXPR)   |
| role           | Checks whether the currently authenticated user (if any) has been associated with any of the specified security roles. Use a comma-delimited list to check for multiple roles. Example:<br><code>&lt;logic:present role="role1,role2,role3"&gt;</code><br><code>code..... &lt;/logic:present&gt;</code><br><br>(RT EXPR) |
| scope          | The bean scope within which to search for the bean named by the <code>name</code> property, or "any scope" if not specified.<br><br>(RT EXPR)  |
| user           | Checks whether the currently authenticated user principal has the specified name.<br><br>(RT EXPR)   |

[Back to top](#)

## redirect - Render an HTTP Redirect

Performs an `HttpServletResponse.sendRedirect()` call to the hyperlink specified by the attributes to this tag. URL rewriting will be applied automatically, to maintain session state in the absence of cookies.

The base URL for this redirect is calculated based on which of the following attributes you specify (you must specify exactly one of them):

- *forward* - Use the value of this attribute as the name of a global `ActionForward` to be looked up, and use the application-relative or context-relative URI found there.
- *href* - Use the value of this attribute unchanged.
- *page* - Use the value of this attribute as an application-relative URI, and generate a server-relative URI by including the context path.

Normally, the redirect you specify with one of the attributes described in the previous paragraph will be left unchanged (other than URL rewriting if necessary). However, there are two ways you can append one or more dynamically defined query parameters to the hyperlink -- specify a single parameter with the `paramId` attribute (and its associated attributes to select the value), or specify the name (and optional `property`) attributes to select a `java.util.Map` bean that contains one or more parameter ids and corresponding values.

To specify a single parameter, use the `paramId` attribute to define the name of the request parameter to be submitted. To specify the corresponding value, use one of the following approaches:

- *Specify only the paramName attribute* - The named JSP bean (optionally scoped by the value of the `paramScope` attribute) must identify a value that can be converted to a `String`.
- *Specify both the paramName and paramProperty attributes* - The specified property getter method will be called on the JSP bean identified by the `paramName` (and optional `paramScope`) attributes, in order to select a value that can be converted to a `String`.

If you prefer to specify a `java.util.Map` that contains all of the request parameters to be added to the hyperlink, use one of the following techniques:

- *Specify only the name attribute* - The named JSP bean (optionally scoped by the value of the `scope` attribute) must identify a `java.util.Map` containing the parameters.
- *Specify both name and property attributes* - The specified property getter method will be called on the bean identified by the name (and optional `scope`) attributes, in order to return the `java.util.Map` containing the parameters.

As the `Map` is processed, the keys are assumed to be the names of query parameters to be appended to the hyperlink. The value associated with each key must be either a `String` or a `String` array representing the parameter value(s). If a `String` array is specified, more than one value for the same query parameter name will be created.

| Attribute Name | Description  |
|----------------|--|
| anchor         | Optional anchor tag ("#xxx") to be added to the generated hyperlink. Specify this value <b>without</b> any "#" character.<br><br>(RT EXPR)   |
| forward        | Logical name of a global <code>ActionForward</code> that contains the actual content-relative URI of the destination of this redirect. This URI may be dynamically modified by the inclusion of query parameters, as described in the tag description. You <b>must</b> specify exactly one of the <code>forward</code> attribute, the <code>href</code> attribute, the <code>linkName</code> attribute, or the <code>page</code> attribute.<br><br>(RT EXPR) |
| href           | The URL to which this redirect will transfer control. This URL may be dynamically modified by the inclusion of query parameters, as described in the tag description. You <b>must</b> specify exactly one of the <code>forward</code> attribute, the <code>href</code> attribute, the <code>linkName</code> attribute, or the <code>page</code> attribute.<br><br>(RT EXPR)  |
| name           | The name of a JSP bean that contains a <code>Map</code> representing the query parameters (if <code>property</code> is not specified), or a JSP bean whose property getter is called to return a <code>Map</code> (if <code>property</code> is specified).<br><br>(RT EXPR)  |
| page           | The context-relative path (beginning with a "/" character) to which this hyperlink will transfer control if activated. This hyperlink may be dynamically modified by the inclusion of query parameters, as described in the tag description. You <b>must</b> specify exactly one of the <code>forward</code> attribute, the <code>href</code> attribute, the <code>linkName</code> attribute, or the <code>page</code> attribute.<br><br>(RT EXPR)           |
| paramId        | The name of the request parameter that will be dynamically added to the generated hyperlink. The corresponding value is defined by the <code>paramName</code> and (optional) <code>paramProperty</code> attributes, optionally scoped by the <code>paramScope</code> attribute<br><br>(RT EXPR)  |
| paramName      | The name of a JSP bean that is a <code>String</code> containing the value for the request parameter named by <code>paramId</code> (if <code>paramProperty</code> is not specified), or a JSP bean whose property getter is called to return a <code>String</code> (if <code>paramProperty</code> is specified). The JSP bean is constrained to the bean scope specified by the <code>paramScope</code> property, if it is specified.<br><br>(RT EXPR)        |

|               |   |
|---------------|---|
| paramProperty | <p>The name of a property of the bean specified by the <code>paramName</code> attribute, whose return value must be a <code>String</code> containing the value of the request parameter (named by the <code>paramId</code> attribute) that will be dynamically added to this hyperlink.</p> <p>(RT EXPR)</p>          |
| paramScope    | <p>The scope within which to search for the bean specified by the <code>paramName</code> attribute. If not specified, all scopes are searched.</p> <p>(RT EXPR)</p>   |
| property      | <p>The name of a property of the bean specified by the <code>name</code> attribute, whose return value must be a <code>java.util.Map</code> containing the query parameters to be added to the hyperlink. You <b>must</b> specify the <code>name</code> attribute if you specify this attribute.</p> <p>(RT EXPR)</p> |
| scope         | <p>The scope within which to search for the bean specified by the <code>name</code> attribute. If not specified, all scopes are searched.</p> <p>(RT EXPR)</p>  |
| transaction   | <p>Set to <code>true</code> if you want the current transaction control token included in the generated URL for this redirect.</p> <p>(RT EXPR)</p>   |

[Back to top](#)

[Overview](#)
[Package](#)
[Class](#)
[Tree](#)
[Deprecated](#)
[Index](#)
[Help](#)
[PREV PACKAGE](#)
[NEXT PACKAGE](#)
[FRAMES](#)
[NO FRAMES](#)
[All Classes](#)

# Package org.apache.struts.taglib.nested

Nested tags & supporting classes extend the base struts tags to allow them to relate to each other in a nested nature.

See:

[Description](#)

## Interface Summary

|                                       |   |
|---------------------------------------|---|
| <a href="#">NestedNameSupport</a>     | This is so that managing classes can tell if a nested tag needs to have its <i>name</i> property set.                                       |
| <a href="#">NestedParentSupport</a>   | This interface is so managing classes of the nested tag can identify a tag as a parent tag that other tags retrieve nested properties from. |
| <a href="#">NestedPropertySupport</a> | This interface is for managing classes of the nested extension, so they can know to set the tag's <i>property</i> property.                 |
| <a href="#">NestedTagSupport</a>      | This is to simply allow managing classes to identify the tags to invoke common methods against them.  |

## Class Summary

|                                       |  |
|---------------------------------------|--|
| <a href="#">NestedPropertyHelper</a>  | A simple helper class that does everything that needs to be done to get the nested tag extension to work.                            |
| <a href="#">NestedPropertyTag</a>     | NestedPropertyTag.   |
| <a href="#">NestedReference</a>       | So that a nested hierarchy can penetrate a dynamic JSP include, this class will hold the details of a bean name and nested property. |
| <a href="#">NestedRootTag</a>         | NestedRootTag.   |
| <a href="#">NestedWriteNestingTag</a> | NestedWriteNestingTag.   |

## Package org.apache.struts.taglib.nested Description

Nested tags & supporting classes extend the base struts tags to allow them to relate to each other in a nested nature. The fundamental logic of the original tags don't change, except in that all references to beans and bean properties will be managed in a nested context.

[ [Introduction](#) ] [ [Foundation Concepts - model](#) ] [ [Foundation Concepts - tags](#) ] [ [Nested Tag List](#) ] [ [The "property" Property](#) ] [ [Implementation Details](#) ]

## Introduction

The nesting extension provides the ability to define a nested object model and efficiently represent and manage that model through JSP's custom tags.

It's written in a layer that extends the current Struts tags, building on their logic and functionality. The layer enables the tags to be aware of the tags which surround them so they can correctly provide the nesting property reference to the Struts system. Struts already supported

properties which use "dot notation" in accessing nested objects and properties.

e.g. `myProperty.childProperty.finalProperty`

Because of this the controller servlet excellently manages this nested model. These tags are about bringing such ease of management to the JSP view of the architecture.

---

## Foundation Concepts - model.

A bean holds a reference to another bean internally, and all access to that bean is handled through the current bean. This act of having one bean's access go through another bean is known as "nesting beans". The first bean is known as the parent bean. The bean which it references, is known as a child bean. The terms "parent" and "child" are commonly used to describe the model's hierarchy.

### A simple example...

Take an object which represents a monkey. The monkey's job is to pick bunches of bananas. On each bunch picked hangs many bananas. If this case was translated to bean objects, the monkey object would have a reference to the bunch objects he picked, and each bunch object would hold a reference to the bananas hanging in the bunch.

### To describe this...

The monkey object is the parent to the bunch object, and the bunch object is a child of the monkey object. The bunch object is parent to its child banana objects, and the child banana objects children of the bunch object. The monkey is higher in the hierarchy than the bananas, and the bananas lower in the hierarchy to the bunches.

One special term to remember is for the most parent class, which is known as the "root" object which starts the hierarchy.

---

## Foundation Concepts - tags.

What the tags provide is an efficient way or representing the above models within JSP tag markup. As a result the tags take on similar relationships to each other. A tag can be the parent of another, and similarly be a child of a parent tag. However the most important part to remember, is that the properties of parent tags define the nested property for the child tags' properties.

One issue which may confuse the new developer, is that even though a tag is a parent tag in a markup sense (the opening tag and closing tag are either side of another tag) does not immediately mean that the child tag will be relative to that tag. Why? Some tags make bad parents. In other words, they're not logical steps in defining a hierarchy.

For example the relationship between the select tag and the options tag. The `html:options` tag "must" be surrounded by a parent `html:select` tag.

eg:

```
<html:select name="myBean" property="mySelectProperty" >
  <html:options name="myBean" property="myOptionsProperty" >
</html:select>
```

In the nested context, this would cause undesired results if the select tag was a parent. The bean reference would become...

```
mySelectProperty.myOptionsProperty
```

...which when translated, Struts would go to the value of the select property and then try to get your options list from that returned value. The extended logic tags are the same. You don't want to extend your properties within the objects the logic tags are evaluating.

To get manage this, the tags in the nested extension are categorised into parent tags and non-parent tags. Those which implement `org.apache.struts.taglib.nested.NestedParentSupport` are classed as parents, and the nested system knows they define levels in the nested hierarchy. Every other tag, does not, and will be skipped if found to be a "markup parent" (like our select tag) and not a "nested parent".

There are also the special case of starting off the hierarchy with a "root" tag. These tags are what the extension requires to provide them with the bean by which the structure will be based on.

## Nested Tag List.

Here's a list of tags in the nested extension, grouped by parent/context functionality. "root", "nested parent", "markup parent" & "basic".

### Root Tags

| markup name | brief description   |
|-------------|---|
| html:form   | For backwards compatibility, you can use the typical form tag to implement your nested hierarchy.                         |
| nested:form | An extension of the above <code>html:form</code> , this is just to provide definition in the nested tag library.          |
| nested:root | When you don't want to configure a form, you can use any bean which is in "scope" by specifying its name within this tag. |

### Nested Parent Tags (Affect the hierarchy)

| markup name    | brief description   |
|----------------|---|
| nested:nest    | This tag executes no logic, simply representing a nesting level for the rest of the markup to relate to.  |
| nested:iterate | Extension of <code>logic:iterate</code> you can use it to iterate through a list, and have all child references nest within the beans returned from this iterated collection. |

### Markup Parent Tags (marked-up like parent tags, but don't affect the hierarchy)

| markup name         | brief description  |
|---------------------|--|
| nested:select       | <code>html:select</code> extension. Provides the logic to render a select box in Html. |
| nested:empty        | <code>logic:empty</code> extension.  |
| nested:notEmpty     | <code>logic:notEmpty</code> extension.   |
| nested:equal        | <code>logic:equal</code> extension.  |
| nested:notEqual     | <code>logic:notEqual</code> extension.   |
| nested:greaterEqual | <code>logic:greaterEqual</code> extension.   |
| nested:greaterThan  | <code>logic:greaterThan</code> extension.  |
| nested:lessEqual    | <code>logic:lessEqual</code> extension.  |
| nested:lessThan     | <code>logic:lessThan</code> extension.   |
| nested:match        | <code>logic:match</code> extension.  |
| nested:notMatch     | <code>logic:notMatch</code> extension.   |
| nested:present      | <code>logic:present</code> extension.  |



|                   |                             |
|-------------------|-----------------------------|
| nested:notPresent | logic:notPresent extension. |
|-------------------|-----------------------------|

**Basic tags** (usually a tag which has no body content)

| markup name              | brief description                 |
|--------------------------|-----------------------------------|
| nested:checkbox          | html:checkbox extension.          |
| nested:hidden            | html:hidden extension.            |
| nested:define            | bean:define extension.            |
| nested:image             | html:image extension.             |
| nested:img               | html:img extension.               |
| nested:link              | html:link extension.              |
| nested:message           | bean:message extension.           |
| nested:multibox          | html:multibox extension.          |
| nested:options           | html:options extension.           |
| nested:optionsCollection | html:optionsCollection extension. |
| nested:password          | html:password extension.          |
| nested:radio             | html:radio extension.             |
| nested:select            | html:select extension.            |
| nested:size              | bean:size extension.              |
| nested:submit            | html:submit extension.            |
| nested:text              | html:text extension.              |
| nested:textarea          | html:textarea extension.          |
| nested:write             | bean:write extension.             |

---

## The relative references and the "property" property.

Use of the "property" property is exactly the same as the original Struts tag with minor addition. Appends the provided property to the nested property of the tags that surround it. You can use additional nesting (use "dot notation") within the provided property as the current struts system allows but there is now a tag which can provide this in a "cleaner" fashion. :)

The one other addition to the "property" property, is the ability to step backwards in the heirarchy in the familiar directory fashion; e.g. "../myPropertyName"

As expected this allows you to step backwards in the nested model to access a higher level in the object tree. The implementation uses the StringTokenizer working off the "/" delimiter and counts the tokens. This was going to be denied, enforcing the ".." fashion, but on consideration, allowed for some easier-to-read naming possibilities.

Consider "propertyOne.propertyTwo.propertyThree.propertyFour". With the current nesting level beneath "propertyFour" you can instead use "two/three/four/anotherProperty" which is easier to understand than "../../anotherProperty". Doesn't sound like much, but makes life easier when traversing large jsp pages for tags defining your object model.

Also implemented is the also familiar directory fashion of a leading "/" to reference from the root of the model and start over. e.g. "/propertyOne". This allow a convenient way to move around a few levels as well as "forking" in the object structure among other flexble approaches to structure.

**Parent References...**

"property" properties, including the relative properties described above, all end up referencing a property of a child bean. For example "/myProperty" will return an object from the "myProperty" of the root bean. The fact that a property is specified means that you are accessing the result of that property. This results in never being able to properly access a parent object itself within its current related context.

Take for example you simply want to print out a list of String objects. In a bean you create a list of them, offer them out to the system via a getter, and you markup using the nested:iterate or logic:iterate tag (both contain the same issues). The only way to get at the object itself is get the iterate tag to declare a scripting variable. With the nested tags you can now simply reference the object using a parent reference of "../" or "this/". Any property ending in the "/" will be treaded as a parent reference. So if you use "parent/" as your property, it will step back one parent and use this block's parent. The special cases to use the parent of the current nested block are "../" or "this/". Not just the iterate tag, this will return the object represented by any nested parent tag.

This allows you to be in a nested tag block and use the custom tags work directly against the parent defined object, indexed or otherwise. So to be in an iterate block, and to print out the String representation of the current iterated object, you can now use...

```
eg:
<nested:iterate property="myItemList" >
  <html:write property="this/" ><br>
</html:iterate>
```

or if you want to print the string value of a parent the other side of the object...

```
eg:
<nested:iterate property="myBeanList" >
  <nested:iterate property="myItemList" >
    <html:write property="beanListObject/" ><br>
  </html:iterate>
</html:iterate>
```


The fact that it didn't use the special cases of "../" or "this/" means that it steps back in the hierarchy as a typical relative reference. This is unlimited the amount of steps you can take back in the hierarchy. For example, to go back three parents your property would be "one/two/three/".

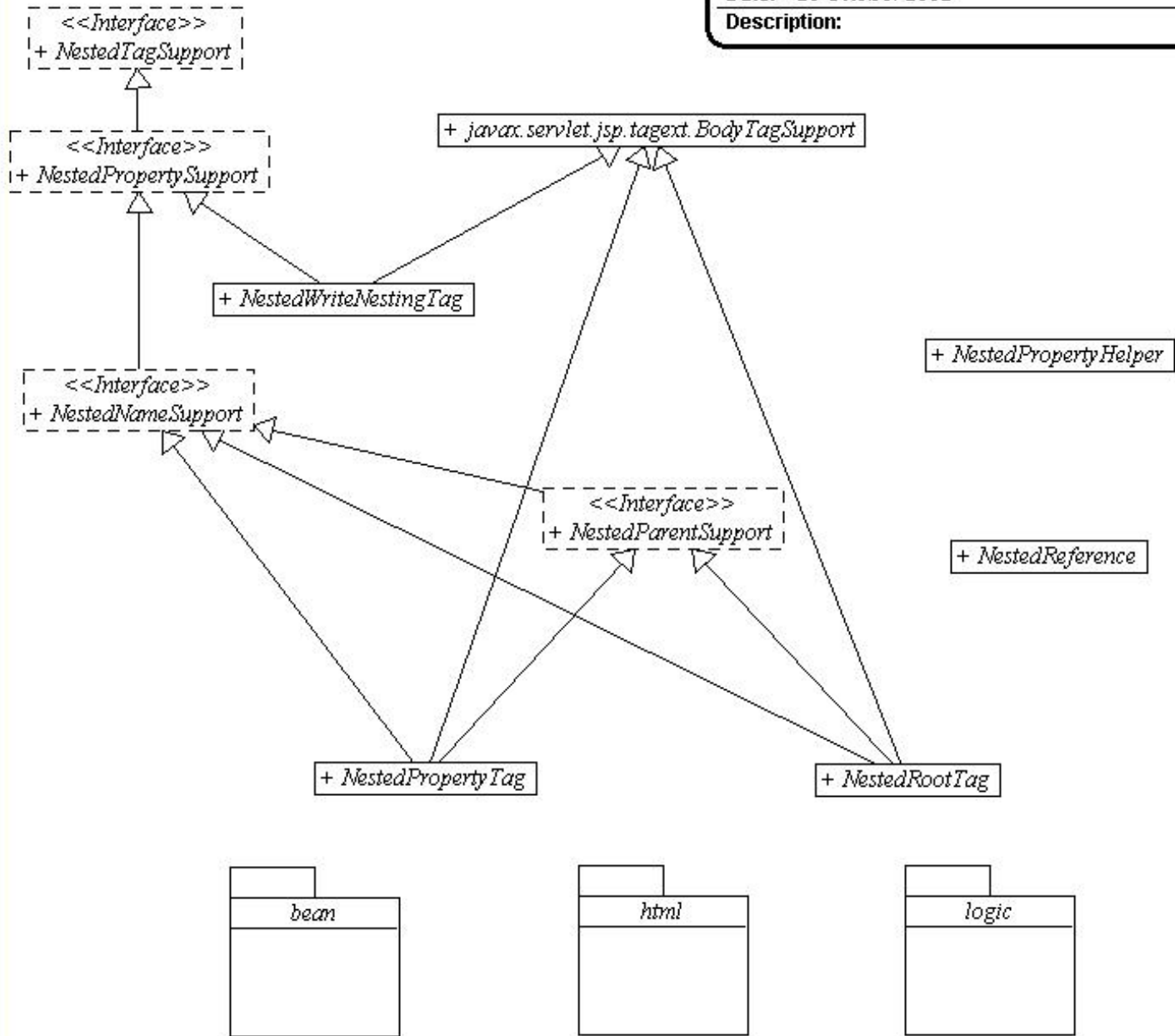
**Note:** The logic identifies the leading "/" and then reads the property from the last index of "/". Resulting in "/three/four/anotherProperty" working the same as "/anotherProperty".

**Note:** If you're busily nesting away, and a parent tag has a leading "/" property, the contained tags will append to this new structure. Handy, but you have to keep it in mind.

**Note:** If you try to reference beneath the level of the nesting, it will simply act like as if a leading "/" property was defined.

**Parent Reference Note:** The only thing to keep in mind with parent references is that you cannot parent reference the root bean. This is because the resulting property would be an empty string. Something that the BeanUtils/PropertyUtils cannot handle (if this is a requirement, you could use a "fake nested property". A getter which simply returns the same bean instance ("this") and simply add an extra nested:nest level at the start of your hierarchy. Works just fine).

|   |                               |                            |
|---|-------------------------------|----------------------------|
|  | <b>Project:</b> <i>Struts</i> | <b>Version:</b> <i>1.1</i> |
| <b>Package:</b> <i>org.apache.struts.taglib.nested</i>                            |                               | <b>Version:</b> <i>1.1</i> |
| <b>Date:</b> <i>29 October 2002</i>   |                               |                            |
| <b>Description:</b>   |                               |                            |



## Implementation Details.

### NestedPropertyHelper

This class embodies all of the logic that runs the nested tagging system. It defines a static method, "setNestedProperties", which the nested tags pass themselves into to have their appropriate properties set.

The tag extensions themselves implement options of three interfaces which define functionality for the various types of nested usage. When traversing the tag hierarchy back up to the root of the structure, these tags define the result of the current tag.

### NestedTagSupport Interface...

This is the base of the interfaces. Simply put, any tag that we need to single out of the standard struts tags for use by the nesting system can implement this or its children.

## NestedPropertySupport Interface...

Tags that implement this interface will have the provided property attribute appended to the parenting nested attribute. This is the heart of the matter.

This is the basic property, and so far all the nested tags support this to have their nested properties written correctly.

## NestedNameSupport Interface...

This interface means that the implementing tag wants to have it's name tag looked after by the nesting system. This is automatic, and the name is written for the tag from the root tag. If the JSP is a form, then it will look to the form tag and get a hold of the bean name that is defined in the struts-config.xml file for the action, otherwise, a nested:root tag must be provided for this means.

This extends the NestedPropertySupport interface as, at time of writing, all tags which used a "name" attribute, required a property attribute in some way to make it useful. This could change, and it's only a small refactoring to make it work for the instance if it's relevant for nesting.

**Note:** At the moment, if the tag implements this interface, the name attribute will be rewritten by the system on all counts. I find it hard to picture a valid requirement for inter-mixing multiple object structures (which distinguishable names would allow) in the one JSP page which couldn't be more efficiently provided by the current nesting model working over the one model. Time may prove this idea wrong.

## ParentTagSupport

This tag identifies for the system those tags which define levels in the nested heirarchy. Namely the "getNestedProperty()" method that yields to calling tags the fully qualified nested property of the parent tag. In the case of a NestedIterator being the parent tag, it will also append the current index reference. e.g. propertyOne.propertyTwo[5]

---

[Overview](#) **[Package](#)** [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV PACKAGE](#) [NEXT PACKAGE](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

---

Copyright © 2000-2002 - Apache Software Foundation



## User Guide

[Table of Contents](#)[Preface](#)[Introduction](#)[Model](#)[Components](#)[View](#)[Components](#)[Controller](#)[Components](#)[Configuration](#)[Release Notes](#)[Installation](#)

## Developer Guides

[Bean Tags](#)[HTML Tags](#)[Logic Tags](#)[Nested Tags](#)[Template Tags](#)[Tiles Tags](#)[Utilities](#)[Validator](#)

## Quick Links

[Welcome](#)[News and Status](#)[Resources](#)[User and](#)[Developer](#)[Guides \\*](#)[FAQs and](#)[HowTos](#)

## Struts Nested Tags

[Since Struts 1.1]

This tag library brings a nested context to the functionality of the Struts custom tag library.

It's written in a layer that extends the current Struts tags, building on their logic and functionality. The layer enables the tags to be aware of the tags which surround them so they can correctly provide the nesting property reference to the Struts system.

### It's all about nesting beans...

A bean holds a reference to another bean internally, and all access to that bean is handled through the current bean. This act of having one bean's access go through another bean is known as "nesting beans". The first bean is known as the parent bean. The bean which it references, is known as a child bean. The terms "parent" and "child" are commonly used to describe the model's hierarchy.

### A simple example...

Take an object which represents a monkey. The monkey's job is to pick bunches of bananas. On each bunch picked hangs many bananas. If this case was translated to bean objects, the monkey object would have a reference to the bunch objects he picked, and each bunch object would hold a reference to the bananas hanging in the bunch.

### To describe this...

The monkey object is the parent to the bunch object, and the bunch object is a child of the monkey object. The bunch object is parent to its child banana objects, and the child banana objects children of the bunch object. The monkey is higher in the hierarchy than the bananas, and the bananas lower in the hierarchy to the bunches.

One special term to remember is for the most parent class, which is known as the "root" object which starts the hierarchy.

Nested tags are all about efficiently managing this style of hierarchy structure within your JSP markup.

**Important Note:** Nearly all these tags extend tags from other libraries to bring their functionality into the nested context. Nesting relies on the tags working against the one bean model, and managing the properties so that they become relative to the properties they are nested within. In doing so, the tags will set the "name" attribute internally (where applicable), and in many cases will rely on the "property" attribute being set so it can be updated internally to become nested. The original tags on occasion provide options that don't use the "name" and "property" attributes. These uses will then fall outside the nested context, and will most likely cause error. To take advantage of these options, markup using the original tag for these cases. For an example see the `<nested:options>` tag.

| Tag Name                           | Description   |
|------------------------------------|---|
| <a href="#">checkbox</a>           | Nested Extension - Render A Checkbox Input Field  |
| <a href="#">define</a>             | Nested Extension - Define a scripting variable based on the value(s) of the specified bean property.  |
| <a href="#">empty</a>              | Nested Extension - Evaluate the nested body content of this tag if the requested variable is either null or an empty string.                  |
| <a href="#">equal</a>              | Nested Extension - Evaluate the nested body content of this tag if the requested variable is equal to the specified value.                    |
| <a href="#">errors</a>             | Nested Extension - Conditionally display a set of accumulated error messages.   |
| <a href="#">file</a>               | Nested Extension - Render A File Select Input Field   |
| <a href="#">form</a>               | Nested Extension - Define An Input Form   |
| <a href="#">greaterEqual</a>       | Nested Extension - Evaluate the nested body content of this tag if the requested variable is greater than or equal to the specified value.    |
| <a href="#">greaterThan</a>        | Nested Extension - Evaluate the nested body content of this tag if the requested variable is greater than the specified value.                |
| <a href="#">hidden</a>             | Nested Extension - Render A Hidden Field  |
| <a href="#">image</a>              | Nested Extension - Render an input tag of type "image"  |
| <a href="#">img</a>                | Nested Extension - Render an HTML "img" tag   |
| <a href="#">iterate</a>            | Nested Extension - Repeat the nested body content of this tag over a specified collection.  |
| <a href="#">lessEqual</a>          | Nested Extension - Evaluate the nested body content of this tag if the requested variable is greater than or equal to the specified value.    |
| <a href="#">lessThan</a>           | Nested Extension - Evaluate the nested body content of this tag if the requested variable is less than the specified value.                   |
| <a href="#">link</a>               | Nested Extension - Render an HTML anchor or hyperlink   |
| <a href="#">match</a>              | Nested Extension - Evaluate the nested body content of this tag if the specified value is an appropriate substring of the requested variable. |
| <a href="#">message</a>            | Nested Extension - Render an internationalized message string to the response.  |
| <a href="#">messages</a>           | Nested Extension - Conditionally display a set of accumulated messages.   |
| <a href="#">messagesNotPresent</a> | Nested Extension - Generate the nested body content of this tag if the specified message is not present in this request.                      |
| <a href="#">messagesPresent</a>    | Nested Extension - Generate the nested body content of this tag if the specified message is present in this request.                          |
| <a href="#">multibox</a>           | Nested Extension - Render A Checkbox Input Field  |
| <a href="#">nest</a>               | Defines a new level of nesting for child tags to reference to   |
| <a href="#">notEmpty</a>           | Nested Extension - Evaluate the nested body content of this tag if the requested variable is neither null nor an empty string.                |
| <a href="#">notEqual</a>           | Nested Extension - Evaluate the nested body content of this tag if the requested variable is not equal to the specified value.                |

|  |   |
|--|---|
| <a href="#"><u>notMatch</u></a>          | Nested Extension - Evaluate the nested body content of this tag if the specified value is not an appropriate substring of the requested variable. |
| <a href="#"><u>notPresent</u></a>        | Nested Extension - Generate the nested body content of this tag if the specified value is not present in this request.                            |
| <a href="#"><u>options</u></a>           | Nested Extension - Render a Collection of Select Options  |
| <a href="#"><u>optionsCollection</u></a> | Nested Extension - Render a Collection of Select Options  |
| <a href="#"><u>password</u></a>          | Nested Extension - Render A Password Input Field  |
| <a href="#"><u>present</u></a>           | Nested Extension - Generate the nested body content of this tag if the specified value is present in this request.                                |
| <a href="#"><u>radio</u></a>             | Nested Extension - Render A Radio Button Input Field  |
| <a href="#"><u>root</u></a>              | To start off a nested hierarchy without the need for a form   |
| <a href="#"><u>select</u></a>            | Nested Extension - Render A Select Element  |
| <a href="#"><u>size</u></a>              | Nested Extension - Define a bean containing the number of elements in a Collection or Map.  |
| <a href="#"><u>submit</u></a>            | Nested Extension - Render A Submit Button   |
| <a href="#"><u>text</u></a>              | Nested Extension - Render An Input Field of Type text   |
| <a href="#"><u>textarea</u></a>          | Nested Extension - Render A Textarea  |
| <a href="#"><u>write</u></a>             | Nested Extension - Render the value of the specified bean property to the current JspWriter.  |
| <a href="#"><u>writeNesting</u></a>      | Writes out the current nesting level, or that defined by a property   |

### **checkbox** - Nested Extension - Render A Checkbox Input Field

This tag is an extension of the [<html:checkbox>](#) tag. Please consult its documentation for information on tag attributes and usage details.

[Back to top](#)

### **define** - Nested Extension - Define a scripting variable based on the value(s) of the specified bean property.

This tag is an extension of the [<bean:define>](#) tag. Please consult its documentation for information on tag attributes and usage details.

[Back to top](#)

### **empty** - Nested Extension - Evaluate the nested body content of this tag if the requested variable is either null or an empty string.

This tag is an extension of the [<logic:empty>](#) tag. Please consult its documentation for information on tag attributes and usage details.

[Back to top](#)

**equal** - Nested Extension - Evaluate the nested body content of this tag if the requested variable is equal to the specified value.

This tag is an extension of the [<logic:equal>](#) tag. Please consult its documentation for information on tag attributes and usage details.

[Back to top](#)

**errors** - Nested Extension - Conditionally display a set of accumulated error messages.

This tag is an extension of the [<html:errors>](#) tag. Please consult its documentation for information on tag attributes and usage details.

[Back to top](#)

**file** - Nested Extension - Render A File Select Input Field

This tag is an extension of the [<html:file>](#) tag. Please consult its documentation for information on tag attributes and usage details.

[Back to top](#)

**form** - Nested Extension - Define An Input Form

This tag is an extension of the [<html:form>](#) tag. Please consult its documentation for information on tag attributes and usage details.

[Back to top](#)

**greaterEqual** - Nested Extension - Evaluate the nested body content of this tag if the requested variable is greater than or equal to the specified value.

This tag is an extension of the [<logic:greaterEqual>](#) tag. Please consult its documentation for information on tag attributes and usage details.

[Back to top](#)

**greaterThan** - Nested Extension - Evaluate the nested body content of this tag if the requested variable is greater than the specified value.

This tag is an extension of the [<logic:greaterThan>](#) tag. Please consult its documentation for information on tag attributes and usage details.

[Back to top](#)



### **hidden** - Nested Extension - Render A Hidden Field

This tag is an extension of the [<html:hidden>](#) tag. Please consult its documentation for information on tag attributes and usage details.

[Back to top](#)

### **image** - Nested Extension - Render an input tag of type "image"

This tag is an extension of the [<html:image>](#) tag. Please consult its documentation for information on tag attributes and usage details.

[Back to top](#)

### **img** - Nested Extension - Render an HTML "img" tag

This tag is an extension of the [<html:img>](#) tag. Please consult its documentation for information on tag attributes and usage details.

[Back to top](#)

### **iterate** - Nested Extension - Repeat the nested body content of this tag over a specified collection.

This tag is an extension of the [<logic:iterate>](#) tag. Please consult its documentation for information on tag attributes and usage details.

[Back to top](#)

### **lessEqual** - Nested Extension - Evaluate the nested body content of this tag if the requested variable is greater than or equal to the specified value.

This tag is an extension of the [<logic:lessEqual>](#) tag. Please consult its documentation for information on tag attributes and usage details.

[Back to top](#)

### **lessThan** - Nested Extension - Evaluate the nested body content of this tag if the requested variable is less than the specified value.

This tag is an extension of the [<logic:lessThan>](#) tag. Please consult its documentation for information on tag attributes and usage details.

[Back to top](#)

### **link** - Nested Extension - Render an HTML anchor or hyperlink

This tag is an extension of the [<html:link>](#) tag. Please consult its documentation for information on tag attributes and usage details.

[Back to top](#)

**match** - Nested Extension - Evaluate the nested body content of this tag if the specified value is an appropriate substring of the requested variable.

This tag is an extension of the [<logic:match>](#) tag. Please consult its documentation for information on tag attributes and usage details.

[Back to top](#)

**message** - Nested Extension - Render an internationalized message string to the response.

This tag is an extension of the [<bean:message>](#) tag. Please consult its documentation for information on tag attributes and usage details.

[Back to top](#)

**messages** - Nested Extension - Conditionally display a set of accumulated messages.

This tag is an extension of the [<html:messages>](#) tag. Please consult its documentation for information on tag attributes and usage details.

[Back to top](#)

**messagesNotPresent** - Nested Extension - Generate the nested body content of this tag if the specified message is not present in this request.

This tag is an extension of the [<logic:messagesNotPresent>](#) tag. Please consult its documentation for information on tag attributes and usage details.

[Back to top](#)

**messagesPresent** - Nested Extension - Generate the nested body content of this tag if the specified message is present in this request.

This tag is an extension of the [<logic:messagesPresent>](#) tag. Please consult its documentation for information on tag attributes and usage details.

[Back to top](#)

**multibox** - Nested Extension - Render A Checkbox Input Field

This tag is an extension of the [<html:multibox>](#) tag. Please consult its documentation for information on tag attributes and usage details.

[Back to top](#)

#### **nest** - Defines a new level of nesting for child tags to reference to

This tag provides a simple method of defining a logical nesting level in the nested hierarchy. It run no explicit logic, is simply a place holder. It also means you can remove the need for explicit setting of level properties in child tags.

Just as the iterate tag provide a parent to other tags, this does the same but there is no logic for iterating or otherwise.

Example...

```
<nested:write property="myNestedLevel.propertyOne" />
<nested:write property="myNestedLevel.propertyTwo" />
<nested:write property="myNestedLevel.propertyThree" />
```

Can instead become...

```
<nested:nest property="myNestedLevel" >
  <nested:write property="propertyOne" />
  <nested:write property="propertyTwo" />
  <nested:write property="propertyThree" />
</nested:nest >
```

| Attribute Name | Description   |
|----------------|---|
| property       | This specifies the property by which this tag and all child tags will be relative to. (RT EXPR) |

[Back to top](#)

#### **notEmpty** - Nested Extension - Evaluate the nested body content of this tag if the requested variable is neither null nor an empty string.

This tag is an extension of the [<logic:notEmpty>](#) tag. Please consult its documentation for information on tag attributes and usage details.

[Back to top](#)

#### **notEqual** - Nested Extension - Evaluate the nested body content of this tag if the requested variable is not equal to the specified value.

This tag is an extension of the [<logic:notEqual>](#) tag. Please consult its documentation for information on tag attributes and usage details.

[Back to top](#)

**notMatch** - Nested Extension - Evaluate the nested body content of this tag if the specified value is not an appropriate substring of the requested variable.

This tag is an extension of the [<logic:notMatch>](#) tag. Please consult its documentation for information on tag attributes and usage details.

[Back to top](#)

**notPresent** - Nested Extension - Generate the nested body content of this tag if the specified value is not present in this request.

This tag is an extension of the [<logic:notPresent>](#) tag. Please consult its documentation for information on tag attributes and usage details.

[Back to top](#)

**options** - Nested Extension - Render a Collection of Select Options

This tag is an extension of the [<html:options>](#) tag. Please consult its documentation for information on tag attributes and usage details.

**Note:** The nested context of this tag relies on the use of the "property" property, and the internal use of the "name" property. The nested tags rely on these properties and will attempt to set them itself. The `<html:options>` tag this tag extended allows other options for the tag which don't use these properties. To take advantage of these options, markup using the `<html:options>` tag instead of the nested tag.

For example, the "collections" option allows you to specify a separate bean reference which itself is a list of objects with properties to access the title and value parts of the html option tag. You can use this in a nested context (the list is a property of a nested bean) by using the nested define tag and the original options tag.

```
<nested:nest property="myNestedLevel" />
  <nested:define property="collectionList" />
  <html:options collection="collectionList"
    property="labelProperty"
    valueProperty="valueProperty" />
</nested:nest >
```

[Back to top](#)

**optionsCollection** - Nested Extension - Render a Collection of Select Options

This tag is an extension of the [<html:optionsCollection>](#) tag. Please consult its documentation for information on tag attributes and usage details.

[Back to top](#)

**password** - Nested Extension - Render A Password Input Field

This tag is an extension of the [<html:password>](#) tag. Please consult its documentation for information on tag attributes and usage details.

[Back to top](#)

**present** - Nested Extension - Generate the nested body content of this tag if the specified value is present in this request.

This tag is an extension of the [<logic:present>](#) tag. Please consult its documentation for information on tag attributes and usage details.

[Back to top](#)

**radio** - Nested Extension - Render A Radio Button Input Field

This tag is an extension of the [<html:radio>](#) tag. Please consult its documentation for information on tag attributes and usage details.

[Back to top](#)

**root** - To start off a nested hierarchy without the need for a form

This tag is provided to allow the nested tags to find a common bean reference without the need for a form and its relative overhead. As long as the name attribute of this tag matches the name of a bean in scope of the JSP (ie: Struts tags can find it via usual means). For example you can load a bean for use with the `jsp:useBean` tag.

The tag can also be used without specifying the name attribute, but this is only in the case that the current JSP is a dynamic include specified in another file. You will not be able to run the tag without a name unless this inclusion is in place. Otherwise the nested tags will not have the bean and property references that they need to provide their logic.

**Note:** The access to a bean via the name attribute takes priority over looking for the reference from other parent tags. So if a name is specified, a bean will have to be there waiting for it. It was made this way so that you could use separate beans within a JSP that itself is an inclusion into another.

| Attribute Name | Description  |
|----------------|--|
| name           | The name of the bean by which all child nested tags will derive their bean reference from. (RT EXPR) |

[Back to top](#)

**select** - Nested Extension - Render A Select Element

This tag is an extension of the [<html:select>](#) tag. Please consult its documentation for information on tag attributes and usage details.

[Back to top](#)

#### **size** - Nested Extension - Define a bean containing the number of elements in a Collection or Map.

This tag is an extension of the [<bean:size>](#) tag. Please consult its documentation for information on tag attributes and usage details.

[Back to top](#)

#### **submit** - Nested Extension - Render A Submit Button

This tag is an extension of the [<html:submit>](#) tag. Please consult its documentation for information on tag attributes and usage details.

[Back to top](#)

#### **text** - Nested Extension - Render An Input Field of Type text

This tag is an extension of the [<html:text>](#) tag. Please consult its documentation for information on tag attributes and usage details.

[Back to top](#)

#### **textarea** - Nested Extension - Render A Textarea

This tag is an extension of the [<html:textarea>](#) tag. Please consult its documentation for information on tag attributes and usage details.

[Back to top](#)

#### **write** - Nested Extension - Render the value of the specified bean property to the current JspWriter.

This tag is an extension of the [<bean:write>](#) tag. Please consult its documentation for information on tag attributes and usage details.

[Back to top](#)

#### **writeNesting** - Writes out the current nesting level, or that defined by a property

This tag provides a way of accessing the nested property reference used by the nested tags. Can expose a scripting variable, or simply write out the value.

| Attribute Name | Description  |
|----------------|--|
| filter         | true/false value, describing to the tag if the result if to be URLEncoded. Helps JavaScript along if the result is required for URL hacking. (RT EXPR) |
| property       | If not supplied, will simply write out as if "/" or "this/" was supplied. (RT EXPR)  |

[Back to top](#)

---

*Copyright (c) 2000-2002, Apache Software Foundation*

Powered by  
**Struts**

# Package org.apache.struts.taglib.template

**Note:** As of Struts 1.1 the template tag library is deprecated in favor of Tiles.

See:

[Description](#)

## Class Summary

|                           |  |
|---------------------------|--|
| <a href="#">GetTag</a>    | <b>Deprecated.</b> <i>Use Tiles instead.</i> |
| <a href="#">InsertTag</a> | <b>Deprecated.</b> <i>Use Tiles instead.</i> |
| <a href="#">PutTag</a>    | <b>Deprecated.</b> <i>Use Tiles instead.</i> |

# Package org.apache.struts.taglib.template Description

**Note:** As of Struts 1.1 the template tag library is deprecated in favor of Tiles.

The "struts-template" tag library contains tags that are useful in creating dynamic JSP templates for pages which share a common format. These templates are best used when it is likely that a layout shared by several pages in your application will change. The functionality provided by these tags is similar to what can be achieved using standard JSP include directive, but are dynamic rather than static.

[\[Introduction\]](#) [\[Template Functionality\]](#) [\[Template UML\]](#) [\[Template Properties\]](#) [\[Template Examples\]](#)

## Introduction

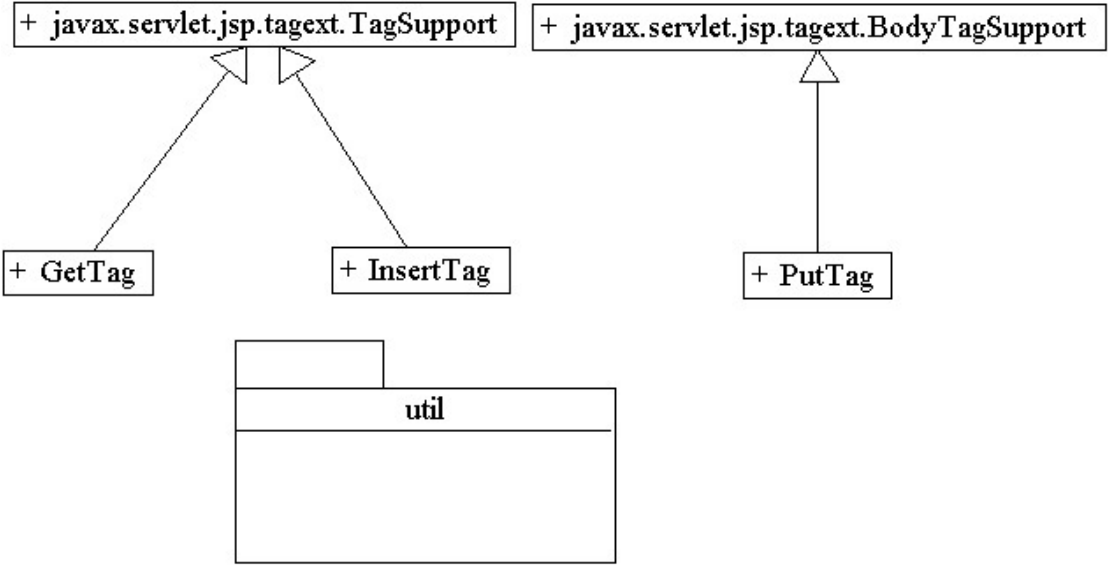
The Template library supplies tags that are useful for creating dynamic JSP templates for pages which share a common format.

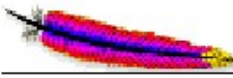
## Template Tag Functionality

Each of the three template tags has a specific, interrelated function:

- [get](#) - Retrieves content from a request scope bean, for use in the template layout.
- [insert](#) - Retrieves (or includes) the specified template file, and then inserts the specified content into the template's layout. By changing the layout defined in the template file, any other file that inserts the template will automatically use the new layout.
- [put](#) - Creates a request scope bean that specifies the content to be used by the get tag. Content can be printed directly or included from a JSP or HTML fil
-





|  |                            |                     |
|--|----------------------------|---------------------|
|  | <b>Date:</b> 31 March 2002 | <b>Version:</b> 1.1 |
| <b>Package:</b> org.apache.struts.taglib.template                                  |                            |                     |
| <b>Description:</b>  |                            |                     |

## Template Tag Properties

The three template tags use very simple attributes :

- get** - Requires a single property, the name of the content to be inserted. This tag is used in the template file to indicate where in the layout to insert the content. This name matches the name property used by a put tag.
- insert** - Requires a single property, the name of the template. This tag is the parent to one or more put tags. The put tags indicate the content to be inserted into the template. The layout of the content is determined by get tags placed in the template.
- put** - Requires a name property, which should match a name used in the template file. The content property indicates the source of the content. The optional direct attribute indicates whether the content should be included or printed directly (as a literal string). The default is false, meaning content is included.

## Template Tag Examples

### A sample template file

```
<%@ taglib uri='/WEB-INF/tlds/struts-template.tld' prefix='template' %>
<html><head><title><template:get name='title' /></title></head>
<body background='graphics/blueAndWhiteBackground.gif'>
<table>
  <tr valign='top'><td><template:get name='sidebar' /></td>
    <td><table>
      <tr><td><template:get name='header' /></td></tr>
      <tr><td><template:get name='content' /></td></tr>
      <tr><td><template:get name='footer' /></td></tr>
    </table>
  </td>
```

```

    </tr>
</table>
</body></html>
<%
/*
"chapterTemplate.jsp"
Display a "sidebar" in a column along the left side of the page.
Display a "header" over the right column.
Display the page "content" below the header.
Display a "footer" at below the content.
If we change the layout of the elements on this page, all pages
inserting this page will also change to use the new layout.
*/
%>

```

---

### A sample JSP using the template

---

```

<%@ taglib uri='/WEB-INF/tlds/struts-template.tld' prefix='template' %>
<template:insert template='/chapterTemplate.jsp'>
  <template:put name='title' content='Templates' direct='true' />
  <template:put name='header' content='/header.html' />
  <template:put name='sidebar' content='/sidebar.jsp' />
  <template:put name='content' content='/introduction.html' />
  <template:put name='footer' content='/footer.html' />
</template:insert>
<%
/*
"introduction.jsp"
Specify template for this page (chapterTemplate.jsp).
The chapterTemplate.jsp defines the layout positions for five
elements: title, header, sidebar, content, and footer.
Specify the source file (html or jsp) for each element.
*/
%>

```

---

### A sample HTML content file ("header.html") used by "introduction.jsp", and others.

---

```

<table>
<tr>
<td><img src='graphics/java.gif' /></td>
<td><img src='graphics/templates.gif' /></td>
</tr>
</table>

```

---

A sample JSP content file ("sidebar.jsp") used by "introduction.jsp", and others.

---

```
<font size='5'><a name="top">Topics</a></font><p>
<table width='145'>
  <tr><td><a href='introduction.jsp'>
    Introduction </a></td></tr>

  <tr><td><a href='using.jsp'>
    Using Templates </a></td></tr>

  <tr><td><a href='optional.jsp'>
    Optional Content </a></td></tr>
  <tr><td><a href='more.jsp'>
    ... and more ...</a></td></tr>
</table></p>

<%
/*
Specify navigational links for this application.
*/
%>
```

---

A sample HTML content file used by "introduction.jsp" only.

---

```
<html>
<head>
<link rel="stylesheet" href="css/templates.css"
charset="ISO-8859-1" type="text/css">
</head>
<body>
<h3 class="ChapTitle">Introduction</h3>
<p class="Paragraph">Window toolkits typically provide a layout mechanism
< ... />
```



## User Guide

[Table of Contents](#)
[Preface](#)
[Introduction](#)
[Model Components](#)
[View Components](#)
[Controller Components](#)
[Configuration](#)
[Release Notes](#)
[Installation](#)

## Developer Guides

[Bean Tags](#)
[HTML Tags](#)
[Logic Tags](#)
[Nested Tags](#)
[Template Tags](#)
[Tiles Tags](#)
[Utilities](#)
[Validator](#)

## Quick Links

[Welcome](#)
[News and Status](#)
[Resources](#)
[User and Developer Guides \\*](#)
[FAQs and HowTos](#)

## Template Tags

This tag library contains three tags: put, get, and insert. Put tags put content into request scope, which is retrieved by a get tag in a different JSP page (the template). That template is included with the insert tag.

| Tag Name                      | Description   |
|-------------------------------|---|
| <a href="#"><u>get</u></a>    | Gets the content from request scope that was put there by a put tag.  |
| <a href="#"><u>insert</u></a> | Inserts (includes, actually) a template. Templates are JSP pages that include parameterized content. That content comes from put tags that are children of insert tags. |
| <a href="#"><u>put</u></a>    | Puts content into request scope.  |

### get - Gets the content from request scope that was put there by a put tag.

Retrieve content from request scope and include it.

| Attribute Name | Description  |
|----------------|--|
| flush          | If set to true, flush the response buffer prior to including the content specified by the name attribute. By default, the response is not flushed. <b>NOTE</b> - this attribute exists only to work around problems on some containers; flushing should never be required. (RT EXPR) |
| name           | The name of the content to get from request scope. (REQUIRED) (RT EXPR)  |
| role           | If the user is in the specified role, the content is retrieved, otherwise, the content is ignored. (RT EXPR)   |

[Back to top](#)

### insert - Inserts (includes, actually) a template. Templates are JSP pages that include parameterized content. That content comes from put tags that are children of insert tags.

This tag's enclosed put tags put content (URIs or text) into request scope. That content is retrieved by get tags in a template (templates are JSP pages). That template is included by this tag's end tag.

| Attribute Name | Description  |
|----------------|--|
| template       | A string representing the URI of a template (a JSP page). (REQUIRED) (RT EXPR) |

[Back to top](#)

### put - Puts content into request scope.

Put content into request scope.

Content can come from attribute content="aContent", or from tag body. In this later case, attribute direct is automatically set to "true".

| Attribute Name | Description  |
|----------------|--|
| content        | Content that's put into request scope. (RT EXPR)   |
| direct         | Determines how content is handled: true means content is printed <i>directly</i> ; false, the default, means content is included. (RT EXPR)    |
| name           | The name of the content. (REQUIRED) (RT EXPR)  |
| role           | If the user is in the specified role, the content is stored for subsequent access by the get tag; otherwise, the content is ignored. (RT EXPR) |

[Back to top](#)

[Overview](#) **[Package](#)** [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)
[PREV PACKAGE](#) [NEXT PACKAGE](#)
[FRAMES](#) [NO FRAMES](#) [All Classes](#)

# Package org.apache.struts.tiles

See:

[Description](#)

## Interface Summary

|  |   |
|--|---|
| <a href="#"><u>AttributeDefinition</u></a>         | Attribute definition used in a component definition.                |
| <a href="#"><u>ComponentDefinitionsFactory</u></a> | <b>Deprecated.</b> <i>Use DefinitionsFactory instead.</i>           |
| <a href="#"><u>Controller</u></a>                  | A controller is a piece of code called before rendering a jsp page. |
| <a href="#"><u>DefinitionsFactory</u></a>          | Tiles Definition factory.   |

## Class Summary

|   |   |
|---|---|
| <a href="#"><u>ActionComponentServlet</u></a>               | Action Servlet to be used with Tiles and Struts 1.0.x.  |
| <a href="#"><u>ActionController</u></a>                     | Struts wrapper implementation of Controller.  |
| <a href="#"><u>ComponentContext</u></a>                     | Component context.  |
| <a href="#"><u>ComponentDefinition</u></a>                  | Definition of a template / component attributes.  |
| <a href="#"><u>ControllerSupport</u></a>                    | Basic implementation of Controller.   |
| <a href="#"><u>DefinitionAttribute</u></a>                  | Attribute representing a Component Definition.  |
| <a href="#"><u>DefinitionNameAttribute</u></a>              | Component attribute.  |
| <a href="#"><u>DefinitionsFactoryConfig</u></a>             | A TilesFactoryConfig object hold configuration attributes for a tile definition factory.  |
| <a href="#"><u>DefinitionsUtil</u></a>                      | <b>Deprecated.</b> <i>Use</i><br><a href="#"><u>TilesUtil.createDefinitionsFactory(ServletContext, DefinitionsFactoryConfig)</u></a>                      |
| <a href="#"><u>DefinitionsUtil.ServletPropertiesMap</u></a> | Inner class.  |
| <a href="#"><u>DirectStringAttribute</u></a>                | Component attribute.  |
| <a href="#"><u>EmptyIterator</u></a>                        |   |
| <a href="#"><u>PathAttribute</u></a>                        | Component attribute.  |
| <a href="#"><u>TilesPlugin</u></a>                          | Tiles Plugin used to initialize Tiles.  |
| <a href="#"><u>TilesRequestProcessor</u></a>                | <b>RequestProcessor</b> contains the processing logic that the Struts controller servlet performs as it receives each servlet request from the container. |

|   |  |
|---|--|
| <a href="#"><u>TilesServlet</u></a>               | A simple servlet initializing and loading Tiles factory.         |
| <a href="#"><u>TilesUtil</u></a>                  | Class containing utilities for Tiles.                            |
| <a href="#"><u>TilesUtilImpl</u></a>              | Default implementation of TilesUtil.                             |
| <a href="#"><u>TilesUtilStrutsImpl</u></a>        | TilesUtil implementation for Struts 1.1 with one single factory. |
| <a href="#"><u>TilesUtilStrutsModulesImpl</u></a> | Implementation of TilesUtil for Struts multi modules.            |
| <a href="#"><u>UntypedAttribute</u></a>           | Common implementation of attribute definition.                   |
| <a href="#"><u>UrlController</u></a>              | Tiles controller including a local URL                           |

## Exception Summary

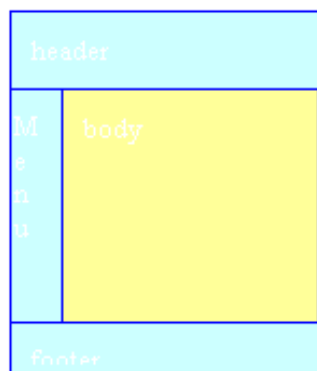
|  |  |
|--|--|
| <a href="#"><u>DefinitionsFactoryException</u></a> | Exception throw when an error occur while factory try to create a new instance mapper. |
| <a href="#"><u>FactoryNotFoundException</u></a>    | Exception throw when instances factory is not found.                                   |
| <a href="#"><u>NoSuchDefinitionException</u></a>   | Exception throw when an instance is not found.   |
| <a href="#"><u>TilesException</u></a>              | Root class of Tiles exception  |

## Package org.apache.struts.tiles Description

The Tiles taglib and framework allows building web pages by assembling reusable pieces of pages, called Tiles. A Tiles is usually a simple JSP page.

## Introduction

The Tiles framework allows building pages by assembling reusable Tiles. As an example, the page in the next figure can be build by assembling a header, a footer, a menu and a body.



Each Tiles (header, menu, body, ...) is a JSP page and can itself be build by assembling other Tiles.

Using Tiles can be compared as using Java methods: You need to define the Tiles (the method body), and then you can "call" this body anywhere you want, passing it some parameters. In Tiles, parameters are called "attributes" in order to avoid confusion with the request parameters.

The Tiles body can be a simple JSP page, a Struts action or any URI pointing to a resource inside the current web site.

Inserting the body, or calling it, is done with the tag `<tiles:insert ...>` anywhere in a JSP page. Insertion can also be done by specifying a *definition name* as the path of a Struts forward or as input, forward or include attributes of a Struts action.

Tiles bodies are used to create layouts, reusable parts, ... Tiles insertions are used to insert Tiles. The same Tiles can be reused several times in the same site, or even in the same page.

Insertion of a Tiles body can be associated to a logical name in what Tiles calls a "definition". A definition contains a logical name, a page used as body and some attribute values. The definition declaration doesn't insert the associated Tiles body. It just associates it with the name. A definition name can be used anywhere insertion of a Tiles body can occur. The associated Tiles body is then inserted with associated attributes.

The definition declarations can be done in JSP pages or in one or more centralized files. A definition can extend another one, overload some attributes, add new attributes ... This allows the declaration of a "master" definition declaring the common layout, header, menu and footer. All other definitions extend this master layout thereby making it possible to change the entire site look & feel simply by changing the master definition.

## Simple Examples

### Insert a JSP page

```
<tiles:insert page="/layouts/commonLayout.jsp" flush="true" />
```

This example inserts the specified page in place of the tag. The page attribute is any valid URL pointing to a resource inside the current site.

### Insert a Tiles passing some attributes

```
<tiles:insert page="/layouts/classicLayout.jsp" flush="true">
  <tiles:put name="title" value="Page Title" />
  <tiles:put name="header" value="/common/header.jsp" />
  <tiles:put name="footer" value="/common/footer.jsp" />
  <tiles:put name="menu" value="/common/menu.jsp" />
  <tiles:put name="body" value="/tiles/mainBody.jsp" />
</tiles:insert>
```

This example inserts the specified page, passing it the attributes. Attributes are stored in a Tiles context which is passed to the inserted pag and can then be accessed by their names.

### Retrieve an attribute value as String

```
<tiles:getAsString name="title" />
```

This example retrieves the value of the attribute "title" and prints it as a String in the current output stream. The method `toString()` is applied on the attribute value, allowing to pass any kind of object as value.



## Insert Tiles referenced by an attribute

```
<tiles:insert attribute='menu' />
```

This inserts the Tiles referenced by the attribute "menu" value. The specified attribute value is first retrieved from current Tiles's context, and then the value is used as a page target to insert.

## Classic Layout

This example is a layout assembling a page in the classic header-footer-menu-body fashion.

```
<%@ taglib uri="/WEB-INF/struts-tiles.tld" prefix="tiles" %>
<HTML>
  <HEAD>
    <link rel=stylesheet href="<%=request.getContextPath()%>/layouts/stylesheet.css"
          type="text/css">
    <title><tiles:getAsString name="title" / ></title>
  </HEAD>
  <body>
    <table border="0" width="100%" cellspacing="5">
    <tr>
      <td colspan="2"><tiles:insert attribute="header" /></td>
    </tr>
    <tr>
      <td width="140" valign="top">
        <tiles:insert attribute='menu' />
      </td>
      <td valign="top" align="left">
        <tiles:insert attribute='body' />
      </td>
    </tr>
    <tr>
      <td colspan="2">
        <tiles:insert attribute="footer" />
      </td>
    </tr>
    </table>
  </body>
</html>
```

The layout is declared in a JSP page (ex: /layouts/classicLayout.jsp). It can be used in conjunction with the tag described in "[Insert a page passing some attributes](#)".

## Definitions

A definition associates a logical name with the URL of a Tiles to be inserted and some attribute values. A definition doesn't insert the Tiles. This is done later using the definition name. A definition name can be inserted as often as you want in your site, making it easy to reuse a Tiles.

A definition can extend another definition and overload some attributes or add new ones. This makes easy factorization of

definitions differing by some attributes. For example, you can define a master definition declaring the main header, menu, footer, and a default title. Then let each of your page definitions extend this master definition and overload the title and the body.

Definitions can be declared in a JSP page, or in one or more centralized files. To enable the definitions from centralized files, you need to initialize the "definitions factory" which will parse the definitions from the files and provide them to the Tiles framework.

## Enabling Definition Factory

To enable Tiles definitions described in one or more files, you need to write these files and to initialize the definition factory.

Initialization is different depending on the Struts version you use, or if you do not use Struts at all.

### Struts1.1

Use the Tiles plug-in to enable Tiles definitions. This plug-in creates the definition factory and passes it a configuration object populated with parameters. Parameters can be specified in the web.xml file or as plug-in parameters. The plug-in first reads parameters from web.xml, and then overloads them with the ones found in the plug-in. All parameters are optional and can be omitted. The plug-in should be declared in each struts-config file:

```
<plug-in className="org.apache.struts.tiles.TilesPlugin" >
  <set-property property="definitions-config"
                value="/WEB-INF/tiles-defs.xml,
                    /WEB-INF/tiles-tests-defs.xml,/WEB-INF/tiles-tutorial-
defs.xml,
                    /WEB-INF/tiles-examples-defs.xml" />
  <set-property property="moduleAware" value="true" />
  <set-property property="definitions-parser-validate" value="true" />
</plug-in>
```

- definitions-config: (optional)
  - Specify configuration file names. There can be several comma separated file names (default: ?? )
- definitions-parser-validate: (optional)
  - Specify if XML parser should validate the Tiles configuration file
    - true : validate. DTD should be specified in file header (default)
    - false : no validation
- moduleAware: (optional)
  - Specify if the Tiles definition factory is module aware. If true (default), there will be one factory for each Struts module. If false, there will be one common factory for all module. In this later case, it is still needed to declare one plugin per module. The factory will be initialized with parameters found in the first initialized plugin (generally the one associated with the default module).
    - true : Tiles framework is module aware
    - false : Tiles framework has one single factory shared among modules (default)
- tilesUtilImplClassname: (optional - for advanced user)
  - Specify The classname of the TilesUtil implementation to use. The specified class should be a subclass of TilesUtilStrutsImpl. This option disable the moduleAware option. Specifying "TilesUtilStrutsImpl" is equivalent to moduleAware = false.

Specifying "TilesUtilStrutsModuleImpl" is equivalent to moduleAware = true.

This option is taken into account only once, when it is first encountered. To avoid problems, it is advice to

specify the same values in all TilesPlugin declaration.

The TilesPlugin class creates one definition factory for each struts module.

If the flag moduleAware is false, only one shared factory is created for all modules. In this later case, the factory is initialized with parameters found in the first plugin. The plugins should be declared in all modules, and the moduleAware flag should be the same for the entire application.

Paths found in Tiles definitions are relative to the main context.

You don't need to specify a TilesRequestProcessor, this is automatically done by the plug-in. If, however, you want to specify your own RequestProcessor, it should extend the TilesRequestProcessor. The plug-in checks this constraint.

## Struts1.0.x

You need to use a special servlet extending the Struts servlet. This is specified in the web.xml file of your application:

```
<servlet>
  <servlet-name>action</servlet-name>
  <servlet-class>org.apache.struts.tiles.ActionComponentServlet</servlet-class>
  <!-- Tiles Servlet parameter
        Specify configuration file names. There can be several comma
        separated file names
    -->
  <init-param>
    <param-name>definitions-config</param-name>
    <param-value>/WEB-INF/tiles-defs.xml</param-value>
  </init-param>
  <!-- Tiles Servlet parameter
        Specify if XML parser should validate the Tiles configuration
file(s).
        true : validate. DTD should be specified in file header.
        false : no validation
    -->
  <init-param>
    <param-name>definitions-parser-validate</param-name>
    <param-value>true</param-value>
  </init-param>
  ...
</servlet>
```

## Without Struts

Tiles can be used without Struts. To initialize the definition factory, you can use the provided servlet. Declare it in the web.xml file of your application:

```
<servlet>
  <servlet-name>action</servlet-name>
  <servlet-class>org.apache.struts.tiles.TilesServlet</servlet-class>
```

```

    <init-param>
    <param-name>definitions-config</param-name>
    <param-value>/WEB-INF/tiles-defs.xml</param-value>
</init-param>
<init-param>
    <param-name>definitions-parser-validate</param-name>
    <param-value>true</param-value>
</init-param>
...

```

The parameters are the same as for Struts1.1 or 1.0.

## Definition File Syntax

The definition file syntax can be found in the [tiles-config\\_1\\_1.dtd file](#).

Following is a simple example:

```

<!DOCTYPE tiles-definitions PUBLIC
    "-//Apache Software Foundation//DTD Tiles Configuration//EN"
    "http://jakarta.apache.org/struts/dtds/tiles-config_1_1.dtd">

<!-- Definitions for Tiles documentation -->
<tiles-definitions>

    <!-- ===== -->
    <!-- Master definition -->
    <!-- ===== -->
    <!-- Main page layout used as a root for other page definitions -->

    <definition name="site.mainLayout" path="/layouts/classicLayout.jsp">
        <put name="title" value="Tiles Blank Site" />
        <put name="header" value="/tiles/common/header.jsp" />
        <put name="menu" value="site.menu.bar" />
        <put name="footer" value="/tiles/common/footer.jsp" />
        <put name="body" value="/tiles/body.jsp" />
    </definition>

    <!-- ===== -->
    <!-- Index page definition -->
    <!-- ===== -->
    <!-- This definition inherits from the main definition.
        It overloads the page title and the body used.
        Use the same mechanism to define new pages sharing common
        properties (here header, menu, footer, layout)
    -->

    <definition name="site.index.page" extends="site.mainLayout" >
        <put name="title" value="Tiles Blank Site Index" />
        <put name="body" value="/tiles/body.jsp" />
    </definition>

```

</tiles-definition>

## Debugging

To debug a page made of Tiles, you can use following advices:

- Check each Tiles separatly. Try to access nested Tiles directly to test if thes work properly.
- Enable Tiles logging. See the commons-logging package help.

---

**Overview** **Package** Class Tree Deprecated Index Help

[PREV PACKAGE](#) [NEXT PACKAGE](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

---

Copyright © 2000-2002 - Apache Software Foundation



## User Guide

- [Table of Contents](#)
- [Preface](#)
- [Introduction](#)
- [Model Components](#)
- [View Components](#)
- [Controller Components](#)
- [Configuration](#)
- [Release Notes](#)
- [Installation](#)

## Developer Guides

- [Bean Tags](#)
- [HTML Tags](#)
- [Logic Tags](#)
- [Nested Tags](#)
- [Template Tags](#)
- [Tiles Tags](#)
- [Utilities](#)
- [Validator](#)

## Quick Links

- [Welcome](#)
- [News and Status](#)
- [Resources](#)
- [User and Developer Guides \\*](#)
- [FAQs and HowTos](#)

## Tiles Tag Library

This tag library provides tiles tags.

Tiles were previously called Components. For historical reasons, names, pages, components and templates are used indifferently to design a tile. Also, a lot of tags and attribute names are left for backward compatibility.

To know more about tags defined in this library, check the associated documentation: tiles-doc.

| Tag Name                                 | Description  |
|--|--|
| <a href="#">add</a>                      | Add an element to the surrounding list. Equivalent to 'put', but for list element.           |
| <a href="#">definition</a>               | Create a tile /component / template definition bean.   |
| <a href="#">get</a>                      | Gets the content from request scope that was put there by a put tag.                         |
| <a href="#">getAsString</a>              | Render the value of the specified tile/component/template attribute to the current JspWriter |
| <a href="#">importAttribute</a>          | Import Tile's attribute in specified context.  |
| <a href="#">initComponentDefinitions</a> | Initialize Tile/Component definitions factory.   |
| <a href="#">insert</a>                   | Insert a tiles/component/template.   |
| <a href="#">put</a>                      | Put an attribute into tile/component/template context.                                       |
| <a href="#">putList</a>                  | Declare a list that will be pass as attribute to tile.                                       |
| <a href="#">useAttribute</a>             | Use attribute value inside page.   |

**add** - Add an element to the surrounding list. Equivalent to 'put', but for list element.

Add an element to the surrounding list. This tag can only be used inside putList tag. Value can come from a direct assignment (value="aValue") or from a bean. One of 'value' or 'beanName' must be present.

| Attribute Name | Description   |
|----------------|---|
| beanName       | Name of the bean used as value. Bean is retrieved from specified context, if any. Otherwise, method pageContext.findAttribute is used. If beanProperty is specified, retrieve value from the corresponding bean property. (RT EXPR) |
| beanProperty   | Bean property name. If specified, value is retrieve from this property. Support nested/indexed properties. (RT EXPR)  |
| beanScope      | Scope into which bean is searched. If not specified, method pageContext.findAttribute is used. Scope can be any JSP scope, 'component', or 'template'. In these two later cases, bean is search in tile/component/template context. |

|         |   |
|---------|---|
| content | Element value. Can be a String or Object. Synonym to value. Attribute added for compatibility with JSP Template. (RT EXPR)  |
| direct  | Determines how content is handled: true means content is printed <i>directly</i> ; false, the default, means content is included. This is another way to specify content type. If 'direct=true' content is 'string', if 'direct=false', content is 'page'. Attribute added for compatibility with JSP Template.   |
| role    | If the user is in the specified role, the tag is taken into account; otherwise, the tag is ignored (skipped).<br>The role isn't taken into account if <add> tag is used in a definition. (RT EXPR)  |
| type    | Specify content type: string, page, template or instance. <ul style="list-style-type: none"> <li>String : Content is printed directly.</li> <li>page   template : Content is included from specified URL. Name is used as an URL.</li> <li>definition : Value denote a definition defined in factory (xml file). Definition will be searched in the inserted tile, in a &lt;insert attribute="attributeName"&gt; tag, where 'attributeName' is the name used for this tag.</li> </ul> <p>If 'type' attribute is not specified, content is 'untyped', unless it comes from a typed bean.</p> |
| value   | Element value. Can be a String or Object.   |

[Back to top](#)

#### definition - Create a tile /component / template definition bean.

Create a tile/component/template definition as a bean. Newly created bean will be saved under specified "id", in the requested "scope". Definition tag has same syntax as `insert` tag. The new definition can extends a definition described in factory (XML file), and overload any previously defined parameters.

| Attribute Name | Description  |
|----------------|--|
| extends        | Name of a parent definition that is used to initialize this new definition. Parent definition is searched in definitions factory. (RT EXPR)  |
| id             | Specifies the name under which the newly created definition bean will be saved. (REQUIRED)   |
| page           | URL of the template / component to insert. Same as "template". (RT EXPR)   |
| role           | Role to check before inserting this definition. If role is not defined for current user, definition is not inserted. Checking is done at insert time, not during definition process. (RT EXPR) |
| scope          | Specifies the variable scope into which the newly defined bean will be created. If not specified, the bean will be created in page scope.  |
| template       | A string representing the URI of a tile/component/template (a JSP page). (RT EXPR)   |

[Back to top](#)

#### get - Gets the content from request scope that was put there by a put tag.

Retrieve content from tile context and include it.  
Take into account the 'type' attribute.

| Attribute Name | Description  |
|----------------|--|
| flush          | True or false. If true, current page out stream is flushed before insertion.   |
| ignore         | If this attribute is set to true, and the attribute specified by the name does not exist, simply return without writing anything. The default value is false, which will cause a runtime exception to be thrown. (RT EXPR) |
| name           | The name of the content to get from tile/component scope. (REQUIRED) (RT EXPR)   |
| role           | If the user is in the specified role, the tag is taken into account; otherwise, the tag is ignored (skipped). (RT EXPR)  |

[Back to top](#)

#### **getAsString** - Render the value of the specified tile/component/template attribute to the current JspWriter

Retrieve the value of the specified tile/component/template attribute property, and render it to the current JspWriter as a String. The usual toString() conversions is applied on found value.  
Throw a JSPException if named value is not found.

| Attribute Name | Description  |
|----------------|--|
| ignore         | If this attribute is set to true, and the attribute specified by the name does not exist, simply return without writing anything. The default value is false, which will cause a runtime exception to be thrown. (RT EXPR) |
| name           | Attribute name. (REQUIRED) (RT EXPR)   |
| role           | If the user is in the specified role, the tag is taken into account; otherwise, the tag is ignored (skipped). (RT EXPR)  |

[Back to top](#)

#### **importAttribute** - Import Tile's attribute in specified context.

Import attribute from tile to requested scope. Attribute name and scope are optional. If not specified, all tile attributes are imported in page scope. Once imported, an attribute can be used as any other beans from jsp contexts.

| Attribute Name | Description   |
|----------------|---|
| ignore         | If this attribute is set to true, and the attribute specified by the name does not exist, simply return without error. The default value is false, which will cause a runtime exception to be thrown. (RT EXPR) |
| name           | Tile's attribute name. If not specified, all attributes are imported. (RT EXPR)   |
| scope          | Scope into which attribute is imported. Default to page.  |

[Back to top](#)



**initComponentDefinitions** - Initialize Tile/Component definitions factory.

Initialize Tile/Components definitions factory.

In order to use Tile/Component definitions factory, you need to initialize the factory. This is generally done in a initializing servlet. In particular, it is done in "ComponentActionServlet" if you use it. If you don't initialize factory in a servlet, you can initialize it using this tag. You need to provide the description file name, and optionally the factory classname. Initialization is done only once, at the first call of this tag. Subsequent calls are ignored (tag checks existence of the factory).

| Attribute Name | Description   |
|----------------|---|
| classname      | If specified, classname of the factory to create and initialized. |
| file           | Definition file name. (REQUIRED)                                  |

[Back to top](#)

**insert** - Insert a tiles/component/template.

Insert a tiles/component/template with the possibility to pass parameters (called attribute). A tile can be seen as a procedure that can take parameters or attributes. `<template:insert>` allows to define these attributes and pass them to the inserted jsp page, called template. Attributes are defined using nested tag `<template:put>` or `<template:putList>`.

You must specify one of this tag attribute :

- `template`, for inserting a tiles/component/template page,
- `component`, for inserting a tiles/component/template page, (same as `template`)
- `page` for inserting a JSP page, (same as `template`)
- `definition`, for inserting a definition from definitions factory
- `attribute`, surrounding tiles's attribute name whose value is used.  
If attribute is associated to 'direct' flag (see `put`), and flag is true, write attribute value (no insertion).
- `name`, to let 'insert' determine the type of entities to insert. In this later case, search is done in this order : definitions, tiles/components/templates, pages.

In fact, Page, component and template, are equivalent as a tile, component or template are jsp page.

**Example :**

```
<template:insert page="/basic/myLayout.jsp" flush="true">
  <template:put name="title" value="My first page" />
  <template:put name="header" value="/common/header.jsp" />
  <template:put name="footer" value="/common/footer.jsp" />
  <template:put name="menu" value="/basic/menu.jsp" />
  <template:put name="body" value="/basic/helloBody.jsp" />
</template:insert>
```

| Attribute Name  | Description  |
|-----------------|--|
| attribute       | Name of an attribute in current tile/component context. Value of this attribute is passed to 'name' (see attribute 'name').  |
| beanName        | Name of the bean used as value. Bean is retrieved from specified context, if any. Otherwise, method <code>pageContext.findAttribute</code> is used. If <code>beanProperty</code> is also specified, retrieve value from the corresponding bean property.<br>If found bean (or property value) is instance of one of Attribute class (Direct, Instance, ...), insertion is done according to the class type. Otherwise, the <code>toString</code> method is called on the bean, and returned String is used as name to insert (see 'name' attribute). (RT EXPR)   |
| beanProperty    | Bean property name. If specified, value is retrieve from this property. Support nested/indexed properties. (RT EXPR)   |
| beanScope       | Scope into which bean is searched. If not specified, method <code>pageContext.findAttribute</code> is used. Scope can be any JSP scope, 'component', or 'template'. In these two later cases, bean is search in tile/component/template context.   |
| component       | Path (relative or absolute to webapps) of the component to insert.<br>'page', 'component' and 'template' are synonyms : they have exactly the same behavior. (RT EXPR)   |
| controllerClass | Class type of a controller called immediately before page is inserted.<br>Controller is used to prepare data to be render by inserted Tile.<br>See also <code>controllerUrl</code><br>Class must implements or extends one of the following : <ul style="list-style-type: none"> <li>• <code>org.apache.struts.tiles.Controller</code></li> <li>• <code>org.apache.struts.tiles.ControllerSupport</code></li> <li>• <code>org.apache.struts.action.Action</code> (wrapper <code>org.apache.struts.action.ActionController</code> is used)</li> </ul><br>See also <code>controllerUrl</code> . Only one of <code>controllerUrl</code> or <code>controllerClass</code> should be used. (RT EXPR) |
| controllerUrl   | Url of a controller called immediately before page is inserted.<br>Url usually denote a Struts action. Controller (action) is used to prepare data to be render by inserted Tile.<br>See also <code>controllerClass</code> . Only one of <code>controllerUrl</code> or <code>controllerClass</code> should be used. (RT EXPR)  |
| definition      | Name of the definition to insert. Definition are defined in a centralized file. For now, only definition from factory can be inserted with this attribute. To insert a definition defined with tag <code>&lt;template:definition&gt;</code> , use <code>beanName=""</code> . (RT EXPR)   |
| flush           | True or false. If true, current page out stream is flushed before insertion.   |
| ignore          | If this attribute is set to true, and the attribute specified by the name does not exist, simply return without writing anything. The default value is false, which will cause a runtime exception to be thrown. (RT EXPR)   |
| name            | Name of an entity to insert. Search is done in this order : definition, attribute, [tile/component/template/page]. (RT EXPR)   |
| page            | Path (relative or absolute to webapps) of the page to insert.<br>'page', 'component' and 'template' are synonyms : they have exactly the same behavior. (RT EXPR)  |

|          |   |
|----------|---|
| role     | If the user is in the specified role, the tag is taken into account; otherwise, the tag is ignored (skipped). (RT EXPR)   |
| template | A string representing the URI of a tile or template (a JSP page).<br>'page', 'component' and 'template' are synonyms : they have exactly the same behavior. (RT EXPR) |

[Back to top](#)

### put - Put an attribute into tile/component/template context.

Define an attribute to pass to tile/component/template. This tag can only be used inside 'insert' or 'definition' tag. Value (or content) is specified using attribute 'value' (or 'content'), or using the tag body. It is also possible to specify the type of the value :

- string : Content is written directly.
- page | template : Content is included from specified URL. Name is used as an URL.
- definition : Content come from specified definition (from factory). Name is used as definition name.

If type is specified, it is taken into account by 'get' or 'insert' inside the inserted tile.

If 'type' attribute is not specified, content is 'untyped', unless it comes from a typed bean.

Note that using 'direct="true"' is equivalent to 'type="string"'.

| Attribute Name | Description   |
|----------------|---|
| beanName       | Name of the bean used as value. Bean is retrieved from specified context, if any. Otherwise, method <code>pageContext.findAttribute</code> is used. If <code>beanProperty</code> is specified, retrieve value from the corresponding bean property. (RT EXPR)   |
| beanProperty   | Bean property name. If specified, value is retrieve from this property. Support nested/indexed properties. (RT EXPR)  |
| beanScope      | Scope into which bean is searched. If not specified, method <code>pageContext.findAttribute</code> is used. Scope can be any JSP scope, 'tile', 'component', or 'template'. In these three later cases, bean is search in tile/component/template context.  |
| content        | Content that's put into tile scope. Synonym to value. Attribute added for compatibility with JSP Template. (RT EXPR)  |
| direct         | Determines how content is handled: true means content is printed <i>directly</i> ; false, the default, means content is included. This is another way to specify content type. If 'direct=true' content is 'string', if 'direct=false', content is 'page'. Attribute added for compatibility with JSP Template. |
| name           | Name of the attribute.  |
| role           | If the user is in the specified role, the tag is taken into account; otherwise, the tag is ignored (skipped). (RT EXPR)   |

|       |  |
|-------|--|
| type  | <p>Specify content type: string, page, template or definition.</p> <ul style="list-style-type: none"> <li>String : Content is printed directly.</li> <li>page   template : Content is included from specified URL. Name is used as an URL.</li> <li>definition : Value is the name of a definition defined in factory (xml file). Definition will be searched in the inserted tile, in a <code>&lt;template:insert attribute="attributeName"&gt;</code> tag, where 'attributeName' is the name used for this tag.</li> </ul> <p>If 'type' attribute is not specified, content is 'untyped', unless it comes from a typed bean.</p> |
| value | Attribute value. Could be a String or an Object. Value can come from a direct assignment (value="aValue") or from a bean. One of 'value' 'content' or 'beanName' must be present. (RT EXPR)  |

[Back to top](#)

#### putList - Declare a list that will be pass as attribute to tile.

Declare a list that will be pass as attribute to tile. List elements are added using the tag 'add'. This tag can only be used inside 'insert' or 'definition' tag.

| Attribute Name | Description                  |
|----------------|------------------------------|
| name           | Name of the list. (REQUIRED) |

[Back to top](#)

#### useAttribute - Use attribute value inside page.

Declare a Java variable, and an attribute in the specified scope, using tile attribute value. Java variable and attribute will have the name specified by 'id', or the original name if not specified.

| Attribute Name | Description   |
|----------------|---|
| classname      | Class of the declared variable.   |
| id             | Declared attribute and variable name.   |
| ignore         | If this attribute is set to true, and the attribute specified by the name does not exist, simply return without error. The default value is false, which will cause a runtime exception to be thrown. (RT EXPR) |
| name           | Tile's attribute name. (REQUIRED) (RT EXPR)   |
| scope          | Scope of the declared attribute. Default to 'page'.   |

[Back to top](#)



[Overview](#)
[Package](#)
[Class](#)
[Tree](#)
[Deprecated](#)
[Index](#)
[Help](#)
[PREV PACKAGE](#)
[NEXT PACKAGE](#)
[FRAMES](#)
[NO FRAMES](#)
[All Classes](#)

# Package org.apache.struts.util

The Utilities package provides a variety of families of classes, to solve problems that are commonly encountered in building web applications.

See:

[Description](#)

## Class Summary

|   |  |
|---|--|
| <a href="#">ErrorMessage</a>                    | <b>Deprecated.</b> <i>Use org.apache.struts.action.ActionErrors instead</i>  |
| <a href="#">GenericDataSource</a>               | <b>Deprecated.</b> <i>Use a BasicDataSource directly, or indirectly acquire a data source provided by your container</i>   |
| <a href="#">ImageButtonBean</a>                 | A simple JavaBean to encapsulate the request parameters sent for an HTML input element of type image.  |
| <a href="#">IteratorAdapter</a>                 | Utility method for converting Enumeration to an Iterator class.  |
| <a href="#">LabelValueBean</a>                  | A simple JavaBean to represent label-value pairs.  |
| <a href="#">MessageResources</a>                | General purpose abstract class that describes an API for retrieving Locale-sensitive messages from underlying resource locations of an unspecified design, and optionally utilizing the MessageFormat class to produce internationalized messages with parametric replacement. |
| <a href="#">MessageResourcesFactory</a>         | Factory for MessageResources instances.  |
| <a href="#">PropertyMessageResources</a>        | Concrete subclass of MessageResources that reads message keys and corresponding strings from named property resources in the same manner that java.util.PropertyResourceBundle does.   |
| <a href="#">PropertyMessageResourcesFactory</a> | Factory for PropertyMessageResources instances.  |
| <a href="#">RequestUtils</a>                    | General purpose utility methods related to processing a servlet request in the Struts controller framework.  |
| <a href="#">ResponseUtils</a>                   | General purpose utility methods related to generating a servlet response in the Struts controller framework.   |
| <a href="#">ServletContextWriter</a>            | A PrintWriter implementation that uses the logging facilities of a javax.servlet.ServletContext to output its results.   |
| <a href="#">StrutsValidator</a>                 | <b>Deprecated.</b> <i>As of Struts 1.1b3, replaced by <a href="#">FieldChecks</a></i>  |
| <a href="#">StrutsValidatorUtil</a>             | <b>Deprecated.</b> <i>As of Struts 1.1b3, replaced by <a href="#">Resources</a></i>  |

## Exception Summary

|                                 |   |
|---------------------------------|---|
| <a href="#">AppException</a>    | <b>Deprecated.</b> <i>use</i> <a href="#">ModuleException</a> |
| <a href="#">ModuleException</a> | Used for specialized exception handling.                      |

## Package org.apache.struts.util Description

The Utilities package provides a variety of families of classes, to solve problems that are commonly encountered in building web applications.

[\[Introduction\]](#) [\[Beans and Properties\]](#) [\[Collection Classes\]](#) [\[JDBC Connection Pool\]](#) [\[Message Resources\]](#)

### Introduction

The Struts Utilities Package offers several families of classes that assist in solving commonly encountered problems when building web applications. Most of the classes in this package do not rely on the controller servlet framework, or the custom tag libraries, so they are also suitable for general Java application programming. The following families are included:

- [Beans and Properties](#) - A family of utility classes for manipulating JavaBeans, as well as getting and setting properties, without hard coding the names or data types of the property getter and setter methods.
- [Collection Classes](#) - A family of specialized classes supporting the Collections API, designed for use in multithread environments where the vast majority of accesses are read only.
- [JDBC Connection Pool](#) - A very simple connection pool that can be utilized in web applications or other environments that need to share a limited number of JDBC Connections across a much larger number of active users.
- [Message Resources](#) - A family of classes that features access to internationalized message strings based on a message key coupled with a `java.util.Locale` object representing a particular user's preferred language.

### Beans and Properties

The `BeanUtils` and `PropertyUtils` utilities are used through out struts including the `IteratorTag`, `WriteTag`. Much of these utilities rely on and make use of Java reflection, to manipulate [Java beans](#). Creating a **valid** Java bean is essential ! Briefly referring to the example class `ProductBean` below would follow these rules :

- The class **must** a null constructor, or no constructor
- It's class declaration **must** be public
- The name for the property say 'value' would have a 'get' method of `getValue()`
- The 'get' and 'set' methods to be visible **must** be public.

- If desired the 'is' prefix can be used in place of 'get' for a boolean.
- Other requirements can be found [here](#).

```
public class ProductBean() {
    private String value;
    public String getvalue() {return this.value}
    public void setvalue(String value) {this.value = value}
}
```

Observing these conventions will avoid unnecessary errors and save time.

This makes it possible to create a JSP page such as:

```
<logic:iterate id="product" name="receivedForm" property="receivedList">
    <bean:write name="product" property="description" />
    <bean:write name="product" property="value" />
</logic:iterate>
```

In this case receiveForm is an ActionForm, with a definition such as

```
public class ReceivedForm extends ActionForm {
    private ProductList productList;
    public void setReceivedList(Enumeration enum) {
        productList = new ProductList(enum,Limits.ARRAY_SIZE_MIN);
    }
    /**
     * Defined so java.bean reflection will see getReceivedList
     * as a getter for receivedList
     */
    public void setReceivedList(ProductList productlist) {

    }

    /**
     * Returns an Array list of ProductBeans.
     */
    public ProductList getReceivedList() {
        return productList;
    };
} //ReceiveForm
```

---

## Collection Classes

### Background



Version 1.2 of the Java 2 Standard Edition (J2SE) introduced a powerful set of collection classes that are generally useful in Java programming, based on the fundamental interfaces `java.util.Collection`, `java.util.List`, `java.util.Map`, and `java.util.Set`. Compared to the collection classes available in JDK 1.1 (principally `java.util.Hashtable` and `java.util.Vector`), the new classes offer much richer functionality as well as the opportunity to improve performance.

The performance increase potential comes from the fact that none of the methods used to access the new collection classes are `synchronized` as were the methods of `Hashtable` and `Vector`. In a single thread application, this means that method calls can execute much more quickly because synchronization is never necessary. In a multiple thread environment, though, it is up to the developer to ensure that any method calls made while another thread is modifying the collection must be synchronized.

There are many cases in multithreaded server environments (such as a web application) where data structures are initialized at application startup time, and are then predominantly accessed in a read-only manner. An example of this is the Struts controller application, which initializes its collection of `ActionMapping` instances (each corresponding to an `<action>` element in the `struts-config.xml` file) at startup time. However, it is legal for an application to dynamically change the set of available mappings while the application is running -- so, to be safe, it would normally be necessary to synchronize access to such collections, even though 99% of those accesses are read only and would not otherwise require synchronization.

To deal with such scenarios, the Struts utility package includes a series of specialized collection classes designed to operate in a multithread environment where the large majority of accesses are read only, without requiring synchronization on every operation, but still protecting against the possibility of runtime modifications to the underlying collection.

## Theory of Operation

Each of the available collection classes operates in one of two modes: *fast* or *slow*. When first created, the collection operates in *slow* mode, which is appropriate for initially populating the contents of the collection. Once the initial population is complete, switch to *fast* mode by calling `setFast(true)` for maximum performance when most accesses are read-only.

When operating in *slow* mode, all methods that access this collection, even read-only methods, are synchronized - resulting in impacts on performance similar to that always performed by the `Hashtable` and `Vector` classes. This mode is appropriate when you are initializing the content of the collection, or when you need to perform a large series of updates.

Using *fast* mode, on the other hand, causes method calls to operate in the following manner:

- Method calls that access information from the collection, but do not modify it, are executed **without** synchronization.
- Method calls that modify the structure of a collection do so by synchronizing, cloning the existing collection instance, modifying the cloned instance, and then replacing the current collection instance.

As you can see, modification operations are **much** more expensive when operating in *fast* mode, but doing things in this way allows read only operations, which should be the vast majority, to operate at maximum speed.

If your collection will **never** be accessed in a multithread environment, you should use one of the standard collection classes instead, without synchronization, for maximum performance.

## Available Collection Classes

The following collection classes, with the ability to operate in either *fast* or *slow* mode, are included:

- [org.apache.struts.util.FastArrayList](#) - Similar in functionality to `java.util.ArrayList`.
- [org.apache.struts.util.FastHashMap](#) - Similar in functionality to `java.util.HashMap`.
- [org.apache.struts.util.FastTreeMap](#) - Similar in functionality to `java.util.TreeMap`.

## JDBC Connection Pool

### Background

A large number of web applications require interaction with a relational database to access or update persistently stored information. In a typical client-server application, each concurrent user opens their own database connection at program initialization, and uses this connection throughout the period of time the application is open.

While this approach can work well in an environment where the number of active users is reasonably fixed, it does not scale well to a web application where the number of simultaneous users could be very large. In addition, open database connections (even when not actively used) do impose some overhead costs, and most web application users (at a given instant) are reviewing the contents of a previously generated page (or typing in their next set of input information), rather than actively accessing the database.

To deal with this situation, several basic strategies are possible:

1. Open a connection on each request, do whatever processing is required, and then close the connection.
2. Open a connection for each user, and store it in the user's session.
3. Share a "pool" of open connections between all of the application's current users.

The first strategy has the virtue of simplicity - you merely need to open a database connection any time you need one, perform the appropriate data accesses and updates, and close the connection. However, it suffers from a major disadvantage: on most databases, establishing a connection can be very time consuming (often requiring multiple seconds of clock time), in order to perform a database transaction that might take milliseconds.

Opening a connection per user, as the second strategy suggests, is similar to the approach taken with client-server applications described earlier. As long as the number of simultaneous users can be controlled at a manageable number (such as with many intranet-based applications), this approach is feasible. However, it becomes unmanageable when the number of users can climb rapidly to very large numbers (as is typical of many Internet-hosted public applications), and still requires more overhead than a strategy that would share a smaller number of connections.

Connection pooling is an implementation of the third strategy. It is based on the assumption that most users of a web application will be interacting locally with the last page that was sent to their browser. The number of users actually performing a request at any given time is usually a very small percentage of the total number of active users, and during request processing is the only time that a database connection is required.

Struts provides a simple connection pool class called `org.apache.struts.util.GenericDataSource`. It allows you to configure a set of connections (with identical connection parameters) to a particular database, using a particular JDBC driver, and then share those connections among a number of simultaneously operating threads (such as the various request threads that are concurrently active in a servlet container). The `GenericDataSource` class implements the `javax.sql.DataSource` interface from the Java Database Connectivity (version 2.0) Standard Extension API, so any programs you use to access it should reference this interface, rather than the class name directly. That way, you can migrate to a more advanced connection pool implementation later, with little or no impact on your application.

For more information about the JDBC 2.0 Standard Extension API, you can download the spec (and the corresponding API classes), from <http://java.sun.com/products/jdbc>. You can also find pointers to a substantial amount of other information about available JDBC drivers, programming tutorials, and so on, at this web address.

## Initializing and Finalizing the Connection Pool

The following instructions show you how to configure the connection pool class and use it, from any Java application. As you will see below, the Struts controller servlet offers you convenient mechanisms to configure one or more connection pools, and make them available to Action classes and JSP pages by storing the connection pool instances as servlet context attributes (in JSP terms, application-scope beans).

To configure a `GenericDataSource` instance, you must first create one:

```
GenericDataSource dataSource =
    new GenericDataSource();
```

Next, you must set the appropriate properties, by calling the corresponding JavaBeans property setter methods provided by this class. (See the Javadoc API for the [GenericDataSource](#) class for more details on the available properties). An example of configuring the connection pool object to a Postgres database might look like this:

```
dataSource.setAutoCommit(false);
dataSource.setDescription("My Database Connection Pool");
dataSource.setDriverClass("org.postgresql.Driver");
dataSource.setMaxCount(4);
dataSource.setMinCount(1);
dataSource.setPassword("mypassword");
dataSource.setUrl("jdbc:postgresql://localhost/mydatabase");
dataSource.setUser("myusername");
```

Finally, you must open ( ) the connection pool. This will establish the initial connections to the database (based on the value you have configured for the `minCount` property). As you use connections from the pool in multiple threads, additional connections (up to the number you specify with the `maxCount` property) will be created as needed.

```
try {
    dataSource.open();
} catch (SQLException e) {
    ... deal with exception ...
}
```

When you are completely through with the connection pool, you can gracefully close all of the currently open database connections by executing

```
try {
    dataSource.close();
} catch (SQLException e) {
    ... deal with exception ...
}
```

## Using the Generic Connection Pool

To access the database from within an application class, you must follow a simple four-step procedure each time you need a connection:

1. Acquire a connection from the connection pool.
2. Perform the database operations required by your application.
3. Cause the last database transaction to be committed or rolled back (commits are required on many databases to ensure that the database operations you just performed are permanently stored or not).
4. "Close" the connection, which returns it to the connection pool for reuse later.

An example code sequence that performs this procedure might look like this:

```
DataSource dataSource = ... acquire reference to dataSource ...
Connection conn = null;
PreparedStatement stmt = null;
ResultSet rs = null;
try {
    conn = dataSource.getConnection();
    stmt = conn.prepareStatement("SELECT cust_id, name FROM customers" +
        " WHERE (last_purchase_date >= ?)" +
        " ORDER BY name");
    stmt.setDate(1, lastPurchaseDate);
    rs = stmt.executeQuery();
    while ((row = rs.next()) != null) {
        ... process this row ...
    }
    rs.close();
    rs = null;
    stmt.close();
    stmt = null;
    conn.commit();
    conn.close();
    conn = null;
} catch (SQLException e) {
    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException f) {
```

```

        ;
    }
    rs = null;
}
if (stmt != null) {
    try {
        stmt.close();
    } catch (SQLException f) {
        ;
    }
    stmt = null;
}
if (conn != null) {
    try {
        conn.rollback();
    } catch (SQLException f) {
        ... deal with exception ...
    }
}
... deal with exception ...
} finally {
    if (conn != null) {
        try {
            conn.close();
        } catch (SQLException f) {
            ... deal with exception ...
        }
        conn = null;
    }
}
}

```

One aspect of the above code example that might surprise developers who have previously used JDBC connections individually is the idea of calling `close()` on the `Connection`. Normally, this call will sever the `Connection`'s underlying link to the database, and render that `Connection` unuseable for any further operations. However, when used in a connection pool environment, the actual `Connection` you receive by calling `getConnection()` is a customized "wrapper" around a real JDBC `Connection` instance. Calling `close()` on this wrapper simply causes this connection to be returned to the pool.

What would happen if your application failed to return a connection to the pool when it was through? As you might expect, that particular connection becomes "lost" to the server, and can never again be used (even though it remains connected to the database throughout the life of the connection pool itself). If this happens repeatedly, you will eventually exhaust the pool of available connections, and application processing will stop.

To avoid this problem, your application logic must ensure that it **ALWAYS** returns allocated connections to the pool, no matter what problems might happen in the interim. The Java language provides one convenient mechanism to achieve this - using a `finally` block, as in the code example above. This is not the only way to ensure that a connection is always returned, but it is very convenient.

## Using Connection Pools with the Struts Controller Servlet

If your application is running underneath the Struts controller servlet (`org.apache.struts.action.ActionServlet`), you can take advantage of the servlet's ability to preconfigure one or more connection pools for you, based on information included in the `struts-config.xml` file. Simply include a section that looks like this:

```
<data-sources>
  <data-source>
    <set-property property="autoCommit"
                  value="false" />
    <set-property property="description"
                  value="Example Data Source Configuration" />
    <set-property property="driverClass"
                  value="org.postgresql.Driver" />
    <set-property property="maxCount"
                  value="4" />
    <set-property property="minCount"
                  value="2" />
    <set-property property="password"
                  value="mypassword" />
    <set-property property="url"
                  value="jdbc:postgresql://localhost/mydatabase" />
    <set-property property="user"
                  value="myusername" />
  </data-source>
</data-sources>
```

After being initialized, the connection pools will be stored as servlet context attributes under the bean name specified by the `key` attribute. If you did not specify a key, the default key is the value of the string constant `Action.DATA_SOURCE_KEY`. Thus, you can access and utilize a connection, from within an `Action` class, like this (for the default data source):

```
DataSource dataSource = (DataSource)
    servlet.getServletContext().getAttribute(Action.DATA_SOURCE_KEY);
conn = dataSource.getConnection();
... perform required functions as in the previous example ...
conn.close();
```

---

## Message Resources

### Background

Modern applications often include the requirement to support multiple languages, for users who prefer to interact in a language other than the default language configured on the server platform. In addition, sentences often need to be constructed, with dynamic content whose placement in the message depends on the standard sentence structure in that particular language.

The standard Java platform includes a family of classes (`java.util.ResourceBundle`) designed to support looking up message strings based on a standard "key". The resource bundle classes automatically access a Java class (or properties file) that is named with a naming convention that includes the Locale to which messages in that class (or file) pertain. However, this selection is based only on the default Locale of the server platform, and cannot be adjusted on a per-user basis as required for an internationalized web application.

Struts includes a family of classes (`org.apache.struts.util.MessageResources`) that extends the basic approach to looking up message strings by key, allowing you to optionally specify a Locale along with the key. In this way, you can build applications that let your users select which Locale they wish to operate within, and then look up messages in that language - using the same message keys no matter what language is selected.

In addition to supporting dynamic selection of a Locale for message lookup, the `MessageResources` family of classes optionally allow you to specify up to four parameter replacement objects, which are used to replace the parameter placeholders "{0}" through "{3}" in the retrieved message. This replacement uses the facilities of the standard Java `java.text.MessageFormat` class, which supports many extended formatting capabilities as well.

For more information about internationalized messages, consult the following resources in your Java Development Kit documentation bundle:

- *Internationalization Info* - General information on Java's standard support for internationalized applications can be found at `<$JAVA_HOME/docs/guide/internat/index.html>`. The "Internationalization Overview" section includes useful information about Locales, localized resources, message formatting, and other relevant topics.
- *Internationalization Tutorial* - The Java Language Tutorial has a comprehensive trail covering internationalization, available at: <http://java.sun.com/docs/books/tutorial/i18n/index.html>.
- *Javadoc APIs* - You will want to consult the Javadoc API documentation for the following standard Java classes:
  - `java.text.MessageFormat`
  - `java.util.ResourceBundle`
  - `java.util.PropertyResourceBundle`
  - `java.util.Properties` - See the documentation for the `load( )` method for the valid syntax of properties files that you prepare.

## Using the Standard MessageResources Implementation

The standard `MessageResources` implementation provided by the Struts library uses Java properties files to initialize message strings, in a manner very similar to that supported by the `java.util.PropertyResourceBundle` class. The following steps are required to use these facilities in your Java application.

First, prepare a Java properties file for each language (or Locale) in which you wish to support your messages. The filenames you use must conform to the naming convention for property resource bundles, as described in the documentation referenced above. Be sure you use the same message keys in each file to identify the same message.

For example, you might prepare files in French, Spanish, and English that contain language-specific versions of the word "Hello". The French file would be named `Messages_fr.properties` and contain the following:

```
hi=Bonjour
```

while the Spanish and English files would be named `Messages_es.properties` and `Messages_en.properties` respectively. The corresponding message string definitions would say `hi=Hola` and `hi=Hello` in these files.

Second, place these properties files into the class path for your application, exactly as you would with class files themselves. The name actually used to load resources will look like a fully qualified Java class name (with appropriate package prefixes), so the file should be nested inside a directory structure that matches the packaging (either in an unpacked directory, or within a JAR file, as appropriate). For example, assume you place directory "foo" on your classpath, and stored the above properties files in directory "foo/com/mycompany/mypackage". (If you were using a JAR file like "foo.jar" instead, the files would be in directory "com/mycompany/mypackage" within the JAR file).

Third, initialize a `MessageResources` object that corresponds to the set of properties files for a particular name, within a particular package. The easiest way to do this is to initialize a variable in your main application class, like this:

```
public static MessageResources messages =
    MessageResources.getMessageResources( "com.mycompany.mypackage.Messages" );
```

Note that the "com.mycompany.mypackage" part of the name matches the package directory into which you placed your properties files, and "Messages" is the filename prefix for the particular family of properties files supported by this `MessageResources` instance. Depending on your development process, you might find it convenient to store all message strings for an entire application in a single properties file family, or to have several families - in Struts, for example, there is a family of properties files for each Java package.

To access a message string with a particular Locale, execute a statement like this:

```
Locale locale = ... select the locale to be used ...
String message = messages.getMessage(locale, "hi");
```

In this case, the variable `message` will contain the message string corresponding to the key "hi", in the language that corresponds to the locale that was selected.

For an example of message formatting with replaceable parameters, assume that the message strings looked like this, instead (only the English version is shown - corresponding changes would be made in the other files):

```
hi=Hello {0}
```

Now, you can personalize the retrieved message like this:

```
Locale locale = ... select the locale to be used ...
String name = "Joe";
String message = messages.getMessage(locale, "hi", name);
```

and the marker "{0}" will have been replaced by the specified name (Joe), no matter which language is in use. See the JavaDoc API documentation for the `java.text.MessageFormat` class for more advanced uses of the parameter replacement mechanism.



## Developing Your Own MessageResources Implementation

In the above example, we were using the default `MessageResources` implementation supplied by Struts, which uses property files to store the message strings. It is also possible to create customized mechanisms to retrieve messages (such as loading them on demand from a database). The steps required are as follows:

- Create a customized subclass of `org.apache.struts.util.MessageResources` that implements message lookup operations as you require.
- Create a customized subclass of `org.apache.struts.util.MessageResourcesFactory` that will create an instance of your custom `MessageResources` class when the `createResources` method is called. Note that the "config" argument to this method can be used to select families of messages in any manner appropriate to your needs - you are not required to emulate the "fully qualified Java class name" approach that is used by the standard `PropertyMessageResourcesFactory` class.
- Tell the `MessageResourcesFactory` class the name of the customized `MessageResourcesFactory` implementation to use when creating new factory instances.
- Create a new factory instance.
- Ask the new factory instance to create a `MessageResources` instance for you.

A code example that illustrates this technique is:

```
MessageResourcesFactory.setFactoryClass("com.mycompany.mypkg.MyFactory");
MessageResourcesFactory factory = MessageResourcesFactory.createFactory();
MessageResources resources =
    factory.createResources("configuration information");
```

Once you have created your custom `MessageResources` instance, you utilize it to access message strings (with or without parameter replacement objects), exactly as we illustrated with the standard implementation in the previous section.

## Using MessageResources With Struts

If your application uses the Struts controller servlet, you can optionally configure Struts to load an application-specific message resources instance for you, and make it available as a servlet context attribute (in JSP terms, an application-scope bean). This mechanism is managed by setting the following servlet initialization parameters in the web application deployment descriptor:

- **application** - The configuration string that will be passed to the `createResources()` method of the message resources factory, in order to identify the family of resources to be supported. If you use the standard message resources factory, this must be the base fully qualified name of the property resources files used to contain these messages, as illustrated above.
- **factory** - Fully qualified Java class name of the `MessageResourcesFactory` to be used. By default, the standard implementation provided by Struts (`org.apache.struts.util.PropertyMessageResourcesFactory`) will be used.

Struts provides several JSP custom tags that assume the existence of a `java.util.Locale` attribute in the user's session, under the key named by the constant string value of `Action.LOCALE_KEY`. Your own application logic can set this attribute at any time, or you can ask Struts to set it automatically (if not already set) based on the `Accept-Language` HTTP header included with the request. There are two mechanisms by which you request Struts to perform

this service:

- To have this service performed on every request submitted to the controller servlet, set the servlet initialization parameter `locale` to the value `true` in the application deployment descriptor.
- To have this service performed by a JSP page when it is accessed directly by a user, utilize a `<form:html ... locale="true" ... />` tag at the top of each page.

---

[Overview](#) **[Package](#)** [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV PACKAGE](#) [NEXT PACKAGE](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

---

Copyright © 2000-2002 - Apache Software Foundation



# Apache Bug Database

Bugzilla version 2.14.2

## Bug List

Fri Jan 17 00:22:41 PST 2003

| <u>ID</u>   | <u>Sev</u> | <u>Pri</u> | <u>Plt</u> | <u>Owner</u>                  | <u>State</u> | <u>Result</u> <u>Summary</u>                              |
|---|------------|------------|------------|-------------------------------|--------------|---|
| <a href="#">15913</a>   | Nor        | Oth        | PC         | struts-dev@jakarta.apache.org | NEW          | Client side validation for integer doesn't work properly. |
| One bug found.  |            |            |            |                               |              |   |
| <a href="#">Query Page</a> <a href="#">Enter New Bug</a> <a href="#">Change columns</a> <a href="#">Edit this query</a> |            |            |            |                               |              |   |

Actions: [New](#) | [Query](#) | bug # | [Reports](#) [New account](#) | [Log in](#)

# Package org.apache.struts.validator

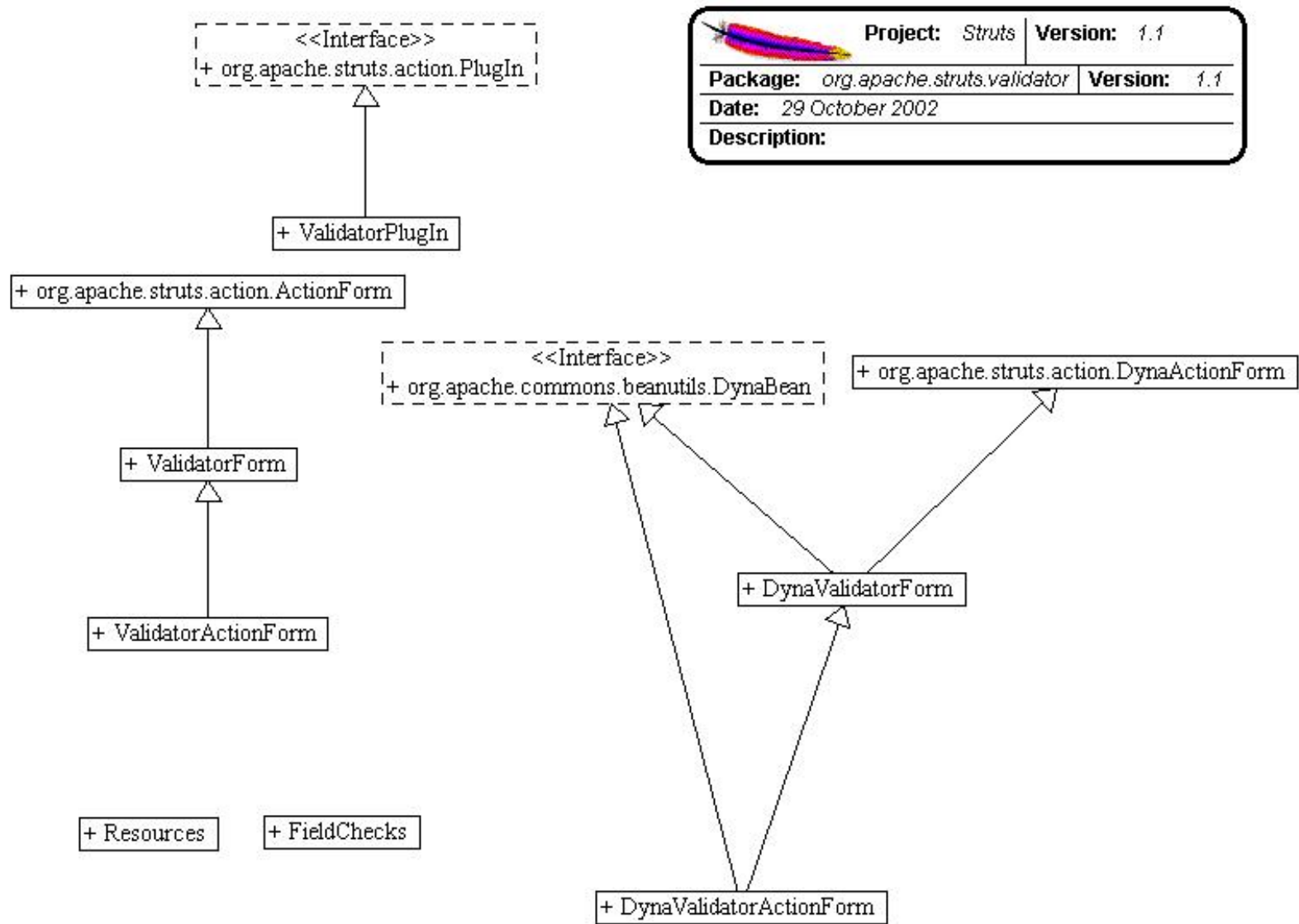
The validator package provides a series of classes to validate `ActionForm` type of input.

See: [Description](#)

| Class Summary                           |  |
|---|--|
| <a href="#">DynaValidatorActionForm</a> | This class extends <b>DynaValidatorForm</b> and provides basic field validation based on an XML file.          |
| <a href="#">DynaValidatorForm</a>       | This class extends <b>DynaActionForm</b> and provides basic field validation based on an XML file.             |
| <a href="#">FieldChecks</a>             | This class contains the default validations that are used in the validator-rules.xml file.                     |
| <a href="#">Resources</a>               | This class helps provides some useful methods for retrieving objects from different scopes of the application. |
| <a href="#">ValidatorActionForm</a>     | This class extends <b>ValidatorForm</b> and provides basic field validation based on an XML file.              |
| <a href="#">ValidatorForm</a>           | This class extends <b>ActionForm</b> and provides basic field validation based on an XML file.                 |
| <a href="#">ValidatorPlugIn</a>         | Loads <code>ValidatorResources</code> based on configuration in the struts-config.xml.                         |

## Package org.apache.struts.validator Description

The validator package provides a series of classes to validate `ActionForm` type of input.



## Package Specification

##### FILL IN ANY SPECS NEEDED BY JAVA COMPATIBILITY KIT #####

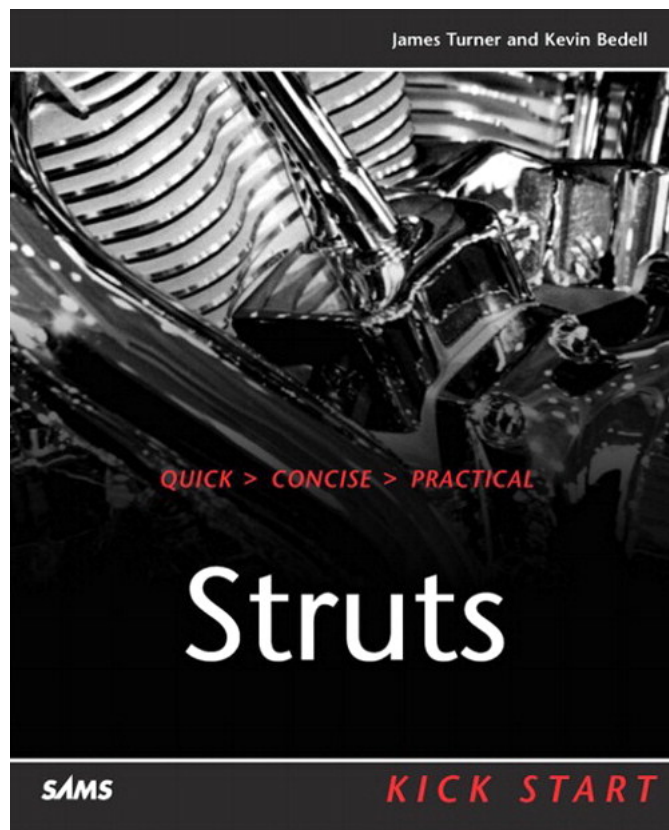
- ##### REFER TO ANY FRAMEMAKER SPECIFICATION HERE #####

## Related Documentation

For overviews, tutorials, examples, guides, and tool documentation, please see:

- ##### REFER TO NON-SPEC DOCUMENTATION HERE #####

Struts 1.1



## January 2003

| Sun | Mon | Tue | Wed                | Thu | Fri | Sat |
|-----|-----|-----|--------------------|-----|-----|-----|
|     |     |     | 1                  | 2   | 3   | 4   |
| 5   | 6   | 7   | 8                  | 9   | 10  | 11  |
| 12  | 13  | 14  | <a href="#">15</a> | 16  | 17  | 18  |
| 19  | 20  | 21  | 22                 | 23  | 24  | 25  |
| 26  | 27  | 28  | 29                 | 30  | 31  |     |

## Site Categories

[Main Page](#)

[Book Errata](#)

[Book News](#)

[News from Kevin](#)

[News from James](#)

[General Struts News](#)

## Search

Search this site:

## January 15, 2003

### Ooops, I Did It Again...

My ever-persuasive editors at SAMS have convinced me to do another Kick Start book, this time covering Java Server Faces. We expect to start writing in the spring for a fall release date.

I've been lucky enough to enlist Craig McClanahan, the spec lead on JSF, to write the forward for the book and provide whatever advice he can fit into his busy schedule.

Posted by blackbear at [03:27 PM](#) | [Comments \(0\)](#)

### Struts 1.1 beta 3 Released

After a lot of effort by the Struts team, the 1.1b3 release has been let loose upon an unsuspecting public. A ton of bug fixes are incorporated into the release, as well as a few new features that snuck in.

You can download the source and binary distributions of the release at <http://www.apache.org/dist/jakarta/struts/>

Now, on to the general release!

Posted by blackbear at [03:24 PM](#) | [Comments \(0\)](#)

## New option for `html:link`

In reference to `html:link` (the description of which begins on page 203 of Struts Kick Start), a new parameter has been added.

Instead of using `<html:link page="/myapp/foo.do">`, you can instead use `<html:link action="/myapp/foo">`, which has the advantage of hiding the actual implementation of action mapping inside the struts-config file, where it belongs.

Posted by blackbear at [03:07 PM](#) | [Comments \(0\)](#)

## December 13, 2002

---

### A Reader Comments

After much anticipation, the book is now available, both from SAMS (with \$8 overnight shipping!!), and Amazon. Readers are already beginning to comment:

From: Ron Day [<mailto:ronday@ronday.cc>]  
Sent: Friday, December 13, 2002 10:55 AM  
To: Struts Users Mailing List  
Subject: RE: [ANNOUNCE] Shameless Plug: Struts Kick Start now available for shipping at Amazon

I'd like to second what Kevin has said.

I have bought and read every book on Struts, in the order that they were published. I got Kick Start this Wednesday. (Sams sent it overnight for \$8.). My three favorite books are Chuck's, Ted's and Kick Start. All three are different, have the personalities and interests of the authors, and offer three perspectives on Struts development and best practices. I would be very hard-pressed to choose just one, but since I love tech books that isn't an issue for me. All three are intermediate to advanced texts (for more intro material, look at "Mastering Struts" ). If I had to give one reason for buying Kick Start, I would say "for the excellent chapters on Struts Tags". The other books do not dwell on the Tags, while KickStart has a detailed chapter on each library.

Ron

Posted by blackbear at [10:55 PM](#) | [Comments \(2\)](#) | [TrackBack \(0\)](#)

### Archives

[January 2003](#)

[December 2002](#)

[November 2002](#)

### Recent Entries

[Ooops, I Did It Again...](#)

[Struts 1.1 beta 3 Released](#)

[New option for `html:link`](#)

[A Reader Comments](#)

[Kevin on Selling Open Source to Management](#)

[Sample chapter from Struts Kick Start available](#)

### Links

[Add Your Links Here](#)

[Syndicate this site \(XML\)](#)

Powered by

[Movable Type 2.51](#)

## December 10, 2002

---

### Kevin on Selling Open Source to Management

Kevin's posted a weblog entry on the O'Reilly site on "Selling Open Source to Management" - here's the URL:

<http://www.oreillynet.com/pub/wlg/2287>

Posted by blackbear at [11:07 PM](#) | [Comments \(0\)](#)